



## 저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

이학 박사 학위논문

# Arithmetics of Ciphertexts under Homomorphic Encryption

(동형암호 기반 암호문 간의  
산술 연산에 관한 연구)

2017년 2월

서울대학교 대학원

수리과학부

김미란

# Arithmetics of Ciphertexts under Homomorphic Encryption

(동형암호 기반 암호문 간의  
산술 연산에 관한 연구)

지도교수 천정희

이 논문을 이학 박사 학위논문으로 제출함

2016년 10월

서울대학교 대학원

수리과학부

김미란

김미란의 이학 박사 학위논문을 인준함

2016년 12월

위 원 장	김	명	환	(인)
부 위 원 장	천	정	희	(인)
위 원	이	향	숙	(인)
위 원	김	명	선	(인)
위 원	Kristin	Lauter		(인)

# Arithmetics of Ciphertexts under Homomorphic Encryption

A dissertation  
submitted in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
to the faculty of the Graduate School of  
Seoul National University

by

Miran Kim

Dissertation Director : Professor Jung Hee Cheon

Department of Mathematical Sciences  
Seoul National University

February 2017

© 2017 Miran Kim

All rights reserved.

# Abstract

## Arithmetics of Ciphertexts under Homomorphic Encryption

Miran Kim

Department of Mathematical Sciences

The Graduate School

Seoul National University

Privacy homomorphism is an important concept for encrypting clear data while allowing one to perform operations on encrypted data without decryption. Although the use of fully homomorphic encryption schemes theoretically allows for the secure evaluation of any function, the evaluation cost is still far from being practical for many functions and no secure solutions have been developed to satisfy the efficiency requirements.

In this thesis, the foundation of our simple framework is a set of optimized circuits for the following operations: equality, greater-than comparison and integer addition. We first focus on the applications of homomorphic encryption for private query processing on encrypted databases. In particular, we construct a unified framework to efficiently and privately process queries with “search” and “compute” operations by applying the underlying circuit primitives.

Since genomic data contains numerous distinguishing features and sensitive personal information, a privacy-preserving genome analysis in a cloud computing environment becomes the major issue in bioinformatics. We present a method to perform the exact edit distance algorithm on encrypted data to

obtain an encrypted result. We also describe how to privately compute the approximate edit distance between encrypted DNA sequences. Finally we create a homomorphic security system for searching a set of biomarkers to encrypted genomes. We propose an efficient method to securely search a matching position with biomarker and extract the information of DNA sequences at the position without complicated computation such as comparison.

**Key words:** homomorphic encryption, privacy query processing, SQL query, exact edit distance, approximate edit distance, biomarkers

**Student Number:** 2012-30071

# Contents

<b>Abstract</b>	<b>i</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Contributions . . . . .	2
1.1.1 Private Database Query Processing . . . . .	2
1.1.2 Secure Genome Analysis . . . . .	3
<b>2 Preliminaries</b>	<b>6</b>
2.1 Practical Homomorphic Encryption . . . . .	6
2.1.1 The BGV-Type Scheme . . . . .	7
2.1.2 The YASHE Scheme . . . . .	8
2.1.3 The Ring-GSW Scheme . . . . .	10
2.2 Human Genome Comparison . . . . .	11
<b>3 Primitive Arithmetic Circuits under Homomorphic Encryption</b>	<b>13</b>
3.1 Binary Arithmetic Circuits . . . . .	14
3.1.1 Equality Circuit . . . . .	14
3.1.2 Greater-Than Comparison Circuit . . . . .	15
3.1.3 Integer Addition Circuit . . . . .	16
3.1.4 Maximum & Minimum Circuits . . . . .	17
3.2 Arithmetic Circuits over the Integers . . . . .	19
3.2.1 Calibrating Binary Circuit Primitives . . . . .	20



## CONTENTS

3.2.2	Arithmetic Circuits over the Integers Based on Fermat's Little Theorem . . . . .	20
3.2.3	Arithmetic Circuits over the Integers Based on Lagrange Interpolation Formula . . . . .	21
<b>4</b>	<b>Private Database Query Processing</b>	<b>23</b>
4.1	General-Purpose Search-and-Compute . . . . .	23
4.1.1	A High-level Overview of Our Approach . . . . .	25
4.1.2	Security Evaluation . . . . .	25
4.2	Applications to Encrypted Databases . . . . .	26
4.2.1	Search Queries . . . . .	26
4.2.2	Search-and-Compute Queries . . . . .	28
4.2.3	Join Queries . . . . .	29
4.3	Performance Improvement . . . . .	30
4.3.1	Larger Message Spaces with Lazy Carry Processing . . . . .	30
4.3.2	Calibrating Circuit Primitives . . . . .	31
4.4	Implementation and Discussion . . . . .	33
4.4.1	Adjusting the Parameters . . . . .	34
4.4.2	Experiments for Search Queries . . . . .	35
4.4.3	Experiments for Search-and-Sum . . . . .	35
4.4.4	Experiments for Search-and-Count . . . . .	38
4.5	Handling Join Query . . . . .	40
<b>5</b>	<b>Secure Genome Analysis Based on Homomorphic Encryption</b>	<b>43</b>
5.1	Exact Edit Distance Algorithm . . . . .	43
5.1.1	Encrypted Edit Distance Algorithm . . . . .	43
5.1.2	Optimizations Based on Block Computation . . . . .	45
5.1.3	Optimization of Encrypted Edit Distance Algorithm Based on Pathfinding Method . . . . .	47
5.1.4	Implementation . . . . .	50

## CONTENTS

5.2	Approximate Edit Distance Algorithm . . . . .	51
5.2.1	Encoding Genomic Data . . . . .	52
5.2.2	Secure DNA Sequence Comparison with Bit-sliced Implementation . . . . .	53
5.2.3	Secure DNA Sequence Comparison with Integer-based Implementation . . . . .	55
5.2.4	Implementation . . . . .	56
5.3	Secure Searching of Biomarkers . . . . .	58
5.3.1	Privacy-Preserving Database Searching and Extraction . . . . .	59
5.3.2	Secure Searching of Biomarkers . . . . .	63
5.3.3	Optimization Techniques . . . . .	64
5.3.4	Implementation . . . . .	66
<b>6</b>	<b>Conclusions</b>	<b>69</b>
	<b>Abstract (in Korean)</b>	<b>77</b>

# Chapter 1

## Introduction

Privacy homomorphism is an important concept for encrypting clear data while allowing one to perform operations on encrypted data without decryption. This concept was first introduced by Rivest *et al.* [RAD78] and, much later, affirmed by Feigenbaum and Merritt's question [FM91]. Is there an encryption function  $\text{Enc}(\cdot)$  such that both  $\text{Enc}(x+y)$  and  $\text{Enc}(x \cdot y)$  can be easily computed from  $\text{Enc}(x)$  and  $\text{Enc}(y)$ ? Since then, there have been a substantial number of studies concerned with solving this problem. However, there had been very little progress made in determining whether such efficient and secure encryption schemes exist until 2009, when Craig Gentry demonstrated the possibility of constructing such an encryption scheme [Gen09].

Roughly Gentry's scheme allows anyone to compute  $\text{Enc}(f(x_1, \dots, x_n))$  from a collection of encrypted data  $\text{Enc}(x_1), \dots, \text{Enc}(x_n)$  for any computable function  $f$  without knowing the actual data. He called this technique a fully homomorphic encryption (FHE) scheme.

While the use of Gentry's scheme and other FHE schemes (e.g., [DGHV10, BV11, BGV12]) allows us to securely evaluate any function in a theoretical sense, the evaluation cost is still far from being practical for many functions. Moreover, the complexity for several important functions has not been determined.

## 1.1 Contributions

### 1.1.1 Private Database Query Processing

Private query processing on encrypted databases allows users to obtain data from encrypted databases in such a way that the users' sensitive data are protected from exposure. Recently, Ada Popa *et al.* proposed CryptDB [PRZB11] and its extension [TKMZ13], which can process general types of database queries using layers of different encryption schemes: deterministic encryption for equality condition queries, order-preserving encryption (OPE) for range queries, and homomorphic encryption (HE) for aggregate queries. The disadvantage of their work is that in the long run, its privacy degrades to the lowest level of data privacy provided by the weakest encryption scheme.

This observation leads to a natural question: *Can we construct a solution to efficiently perform such a database query without maintaining multiple contexts of encryption?* At first glance, HE schemes appear to perfectly satisfy the requirement of processing queries on encrypted databases within a single encryption context. However, no solutions exist for *expressing* and *processing* various queries on *fully* encrypted databases in an *efficient* manner.

Our main results can be summarized as follows: (i) A general framework for private query evaluation, (ii) Optimization of circuits and their applications to compact expressions of queries, (iii) Further enhancement of the performance of query evaluation. We provide a common platform to allow database users to work on a *single underlying cryptosystem*, to represent their queries as functions in a conceptually simple manner, and to efficiently perform these functions on fully encrypted databases. In order to optimize the circuit primitives, we make extensive use of single-instruction-multiple-data (SIMD) techniques to move data across plaintext slots. In general, SIMD technology allows basic operations to be performed on several data elements in parallel. In contrast, our proposal operates on packed ciphertexts of several data elements and thus enables the efficiency of the basic operations of the circuit primitives to be improved. Furthermore, we find that all circuit prim-

## CHAPTER 1. INTRODUCTION

itives have  $O(\log \mu)$  depth for  $\mu$ -bit data. HE schemes typically use  $\mathbb{Z}_2$  as a message space, meaning that their encryption algorithm encrypts each bit of a message. Although our circuit primitives function efficiently on bit encryptions, we can achieve further improvements by adopting a large integer ring (*e.g.*,  $\mathbb{Z}_{2^t}$ ), particularly in the case of *computing* on encrypted numerical data. Even for an integer of  $k$  bits with  $k > t$ , addition can be performed using degree 1 circuits by processing lazy carry operations. Although this rectification requires modification of our circuit primitives, we can preserve their optimality through SIMD operations. In other words, search-and-compute queries can be processed using only  $O(\log \mu)$ -depth circuits. Finally we perform comprehensive experiments to evaluate the performance of various queries using our techniques from both a theoretical and a practical perspective.

### 1.1.2 Secure Genome Analysis

We consider the potential for using homomorphic encryption to protect privacy in genomic computations. There are many projects to collect DNA information from participants in order to discover genomic sequences associated with disease susceptibility. The Personal Genome Project (PGP) displays genotypic and phenotypic information in a public database and the HapMap Project has developed a public repository of genome sequences, which means that genomic data has become publicly accessible. However, even anonymized genomic data can leak significant information about the participants (see for example [GMG<sup>+</sup>13, SAW13, EN14]). Even if DNA sequences are not associated with explicit identifiers such as name, sex, date of birth, or address, one can recover such personal data using re-identification methods: genotype-phenotype inference [MS01], location-visit patterns [MS04], family structure [HAHT13], and dictionary attacks. Thus DNA sequences are sensitive and valuable enough that we should not reveal our own sequences even when performing sequence analysis.

In this thesis, we focus on the well-known exact edit distance algorithm by Wagner and Fisher, which measures the dissimilarity of two strings. We

## CHAPTER 1. INTRODUCTION

show that the algorithm can be implemented on two encrypted sequences of lengths  $n$  and  $m$  with a somewhat homomorphic scheme of depth  $O((n + m) \log(\log(n + m)))$  in  $O(nm \log(n + m))$  homomorphic computations. Moreover, we introduce an optimization technique to reduce the depth: Divide the edit distance matrix into sub-blocks of size- $(\tau + 1)$  and solve the edit distance problem in each block. We can compute each of them diagonally, consuming  $O(\tau)$  levels in one diagonal-round. Namely, evaluating the circuits in each cell can be processed by a somewhat homomorphic encryption of a constant depth. In particular, in the case of  $n = m$ , we improved the efficiency of our protocol with the pathfinding method. The improved protocol can reduce the circuit depth and computational costs with accurate edit distance outputs.

We also propose efficient evaluation algorithms to compute the approximate edit distance on encrypted data which can be used in sequence alignment and gene finding. We adapt these methods to the practical HE schemes – BGV-type scheme [GHS12] by Gentry, Halevi and Smart and YASHE scheme [BLLN13] by Bos, Lauter, Loftus and Naehrig. We also compare the performance of the two encryption schemes in these contexts.

Finally we suggest an efficient method to securely search biomarkers using hybrid Ring-GSW homomorphic encryption scheme. The important feature of our method is to encode a genomic database as a single element of polynomial ring. The searching operation in a database is done by a single multiplication with the encoded query gene, thus the information of DNA sequences at the matched position (*e.g.*, chromosome and position) is moved to the constant term of output polynomial. Finally we perform the extraction procedure to obtain the DNA sequence and compare it with the query DNA information in plaintext. Our method has the advantage over previous methods in parameter size and computational complexity since it requires only a single multiplication. Our implementation shows that the computation of *search-and-extract* takes about 3.9 seconds while extracting respective two single nucleotide polymorphisms of reference and alternate (*i.e.*, non-reference) sequences in a database of size 4M.

## CHAPTER 1. INTRODUCTION

**List of Papers.** This thesis contains results of private database query processing that were obtained jointly with Jung Hee Cheon and Myungsun Kim [CKK15], which was presented in FC workshop 2015, and a forthcoming work [CKK16]. The first protocol of secure computation of edit distance originally appeared in [CKL15], and it was improved the performance in the forthcoming works [KL15, ZDW<sup>+</sup>15]. It also contains the result of secure searching of biomarkers under homomorphic encryption by [KSC17].

- [CKK15] Jung Hee Cheon, Miran Kim, and Myungsun Kim: Search and-compute on encrypted data. In International Conference on Financial Cryptography and Data Security, pages 142-159, Springer, 2015.
- [CKK16] Jung Hee Cheon, Miran Kim, and Myungsun Kim: Optimized search-and-compute circuits and their application to query evaluation on encrypted data. IEEE Transactions on Information Forensics and Security, 11(1):188-199, 2016.
- [CKL15] Jung Hee Cheon, Miran Kim, and Kristin Lauter. Homomorphic computation of edit distance. In International Conference on Financial Cryptography and Data Security, pages 194-212. Springer, 2015.
- [KL15] Miran Kim and Kristin Lauter. Private genome analysis through homomorphic encryption. BMC medical informatics and decision making, 15(Suppl 5):S3, 2015.
- [ZDW<sup>+</sup>15] Yuchen Zhang, Wenrui Dai, Shuang Wang, Miran Kim, Kristin Lauter, Jun Sakuma, Hongkai Xiong, and Xiaoqian Jiang: SECRET: Secure Edit distance computation over homomorphic Encrypted data. Proceedings of the 5th Annual Translational Bioinformatics Conference Tokyo, Japan, 2015.
- [KSC17] Miran Kim, Yongsoo Song, Jung Hee Cheon: Secure searching of biomarkers using hybrid GSW encryption scheme, preprint.

# Chapter 2

## Preliminaries

We will use bold-face lower-case letter  $\mathbf{a}, \mathbf{b} \dots$  to denote column vectors over  $\mathbb{Z}$  or any other ring  $\mathcal{R}$ , and boldface upper-case letters  $\mathbf{A}, \mathbf{B} \dots$  for matrices. The product symbol  $\cdot$  will be used for both inner product of two column vectors, and for matrix product, to be interpreted as the only applicable one.

### 2.1 Practical Homomorphic Encryption

Fully Homomorphic cryptosystems allow us to homomorphically evaluate any arithmetic circuit without decryption. However, the noise of the resulting ciphertext grows during homomorphic evaluations, slightly with addition but substantially with multiplication. For efficiency reasons for tasks which are known in advance, we use a more practical *Somewhat Homomorphic Encryption* (SHE) scheme, which evaluates functions up to a certain complexity. In particular, two techniques are used for noise management of SHE: one is the *modulus-switching* technique introduced by Brakerski, Gentry and Vaikuntanathan [BGV12], which scales down a ciphertext during every multiplication operation and reduces the noise by its scaling factor. The other is a *scale-invariant* technique proposed by Brakerski such that the same modulus is used throughout the evaluation process [Bra12].

Let us denote by  $[\cdot]_q$  the reduction modulo  $q$  into the interval  $(-q/2, q/2] \cap$



## CHAPTER 2. PRELIMINARIES

$\mathbb{Z}$  of the integer or integer polynomial (coefficient-wise). For a security parameter  $\lambda$ , we choose an integer  $m = m(\lambda)$  that defines the  $m$ -th cyclotomic polynomial  $\Phi_m(x)$ . For a polynomial ring  $\mathcal{R} = \mathbb{Z}[x]/(\Phi_m(x))$ , set the plaintext space to  $\mathcal{R}_t := \mathcal{R}/t\mathcal{R}$  for some fixed  $t \geq 2$  and the ciphertext space to  $\mathcal{R}_q := \mathcal{R}/q\mathcal{R}$  for an integer  $q = q(\lambda)$ . Let  $\chi = \chi(\lambda)$  denote a noise distribution over the ring  $R$ . We use the standard notation  $a \leftarrow \mathcal{D}$  to denote that  $a$  is chosen from the distribution  $\mathcal{D}$ .

The ring-based homomorphic encryption schemes are based on the (decisional) Ring Learning With Errors (RLWE) assumption, which was first introduced by Lyubashevsky, Peikert and Regev [LPR10]. The assumption is that it is infeasible to distinguish the following two distributions. The first distribution consists of pairs  $(a_i, u_i)$ , where  $a_i, u_i \leftarrow \mathcal{R}_q$  uniformly at random. The second distribution consists of pairs of the form  $(a_i, b_i) = (a_i, a_i s + e_i)$  where  $a_i \leftarrow \mathcal{R}_q$  drawn uniformly and  $s, e_i \leftarrow \chi$ .

### 2.1.1 The BGV-Type Scheme

Gentry, Halevi and Smart [GHS12] constructed an efficient BGV-type SHE scheme. Note that we can generate RLWE samples as  $(a_i, a_i s + t e_i)$  where  $t$  and  $q$  are relatively prime. To improve efficiency for HE, they use very sparse secret keys  $s$  with coefficients sampled from  $\{-1, 0, 1\}$ . Here is the somewhat homomorphic encryption scheme of [GHS12]:

- **BGV.ParamsGen( $\lambda$ )**: Given the security parameter  $\lambda$ , choose an odd integer  $m$ , a chain of moduli  $q_0 < q_1 < \dots < q_{L-1} = q$ , a plaintext modulus  $t$  with  $1 < t < q_0$ , and discrete Gaussian distribution  $\chi_{err}$ . Output  $(m, \{q_i\}, t, \chi_{err})$ .
- **BGV.KeyGen( $m, \{q_i\}, t, \chi_{err}$ )**: On the input parameters, choose a random  $s$  from  $\{0, \pm 1\}^{\phi(m)}$  and generate an RLWE instance  $(a, b) = (a, [as + te]_q)$  for  $e \leftarrow \chi_{err}$ . We set the key pair:  $(pk, sk) = ((a, b), s)$  with an evaluation key  $evk \in R_{P, q_{L-2}}^2$  for a large integer  $P$ .

## CHAPTER 2. PRELIMINARIES

- **BGV.Enc**( $m, \text{pk}$ ): To encrypt  $m \in \mathcal{R}_t$ , choose a small polynomial  $v$  and two Gaussian polynomials  $e_0, e_1$  over  $\mathcal{R}_q$  and output the ciphertext  $\text{ct} = (c_0, c_1) = (m, 0) + (bv + te_0, av + te_1) \in \mathcal{R}_q^2$ .
- **BGV.Dec**( $\text{ct}, \text{sk}$ ): Given a ciphertext  $\text{ct} = (c_0, c_1)$  at level  $l$ , output  $[c_0 - s \cdot c_1]_{q_l} \bmod t$  where the polynomial  $[c_0 - s \cdot c_1]_{q_l}$  is called the *noise* in the ciphertext  $\text{ct}$ .
- **BGV.Add**( $\text{ct}, \text{ct}'$ )/**BGV.Mult**( $\text{ct}, \text{ct}', \text{evk}$ ): Given two ciphertexts  $\text{ct} = (c_0, c_1)$  and  $\text{ct}' = (c'_0, c'_1)$  at level  $l$ , the homomorphic addition is computed by  $\text{ct}_{\text{add}} = ([c_0 + c'_0]_{q_l}, [c_1 + c'_1]_{q_l})$ . The homomorphic multiplication is computed by  $\text{ct}_{\text{mult}} = \text{SwitchKey}(\text{ct} * \text{ct}', \text{evk})$  where  $\text{ct} * \text{ct}' = ([c_0 c'_0]_{q_l}, [c_0 c'_1 + c_1 c'_0]_{q_l}, [c_1 c'_1]_{q_l})$  and the key switching function **SwitchKey** is used to reduce the size of ciphertexts to two ring elements. We also apply modulus switching from  $q_i$  to  $q_{i-1}$  in order to reduce the noise. If we reach the smallest modulus  $q_0$ , we can no longer compute on ciphertexts.

Smart and Vercauteren [SV14] observed that the polynomial ring  $R_t$  is isomorphic to  $\prod_{i=1}^{\ell} \mathbb{Z}_t[x]/f_i(x)$  if  $\Phi_m(x)$  factors modulo  $t$  into  $\ell$  irreducible factors  $f_i(x)$  of the same degree. Namely, a plaintext polynomial  $m$  can be considered as a vector of  $\ell$  small polynomials,  $m \bmod f_i$ , called *plaintext slots*. We can also transform the plaintext vector  $(m_1, \dots, m_r) \in \prod_{i=1}^{\ell} \mathbb{Z}_t[x]/f_i(x)$  to an element  $m \in \mathcal{R}_t$  using the polynomial Chinese Remainder Theorem (*i.e.*,  $m = \text{CRT}(m_1, \dots, m_r)$ ). In particular, it is possible to add and multiply on the slots: if  $m, m' \in R_t$  encode  $(m_1, \dots, m_{\ell})$  and  $(m'_1, \dots, m'_{\ell})$  respectively, then we see that  $m + m' = m_i + m'_i \bmod f_i$  and  $m \cdot m' = m_i \cdot m'_i \bmod f_i$ . This technique was adapted to the BGV scheme.

### 2.1.2 The YASHE Scheme

The practical SHE scheme, **YASHE**, was proposed in [BLLN13] based on combining ideas from [Bra12, SS11, LATV12]. The security of this scheme is based on the hardness of the RLWE assumption similar to the one for **BGV**.

## CHAPTER 2. PRELIMINARIES

It also relies on the Decisional Small Polynomial Ratio (DSPR) assumption which was introduced by Lopez-Alt, Tromer, and Vaikuntanathan [LATV12]. Let  $t \in \mathcal{R}_q^\times$  be invertible in  $\mathcal{R}_q$ ,  $y_i \in \mathcal{R}_q$  and  $z_i = y_i/t \pmod{q}$  for  $i = 1, 2$ . For  $z \in \mathcal{R}_q$ , we define  $\chi_z = \chi + z$  to be the distribution shifted by  $z$ . The assumption is that it is hard to distinguish elements of the form  $h = a/b$ , where  $a \leftarrow y_1 + t\chi_{z_1}, b \leftarrow y_2 + t\chi_{z_2}$ , from elements drawn uniformly from  $\mathcal{R}_q$ . The YASHE scheme consists of the following algorithms.

- **YASHE.ParamsGen( $\lambda$ )**: Given the security parameter  $\lambda$ , choose  $m$  to be a power of two, modulus  $q$  and  $t$  with  $1 < t < q$ , truncated discrete Gaussian distribution  $\chi_{err}$  on  $\mathcal{R}$  such that the coefficients of the polynomial are selected in the range  $[-B(\lambda), B(\lambda)]$ , and an integer base  $\omega > 1$ . Output  $(m, q, t, \chi_{err}, \omega)$ .
- **YASHE.KeyGen( $m, q, t, \chi_{err}, \omega$ )**: On the input parameters, sample  $f', g \leftarrow \{0, \pm 1\}^{\phi(m)}$  and set  $f = [tf' + 1]_q$ . If  $f$  is not invertible modulo  $q$ , choose a new  $f'$  and compute the inverse  $f^{-1} \in R$  of  $f$  modulo  $q$  and set  $h = [tgf^{-1}]_q$ . Let  $\ell_{\omega, q} = \lfloor \log_\omega(q) \rfloor + 1$  and define  $P_{\omega, q}(a) = ([a\omega^i]_q)_{i=0}^{\ell_{\omega, q}-1}$ . Sample  $\mathbf{e}, \mathbf{s} \leftarrow \chi_{err}^{\ell_{\omega, q}}$  and compute  $\gamma = [P_{\omega, q}(f) + \mathbf{e} + h\mathbf{s}]_q \in \mathcal{R}_q^{\ell_{\omega, q}}$ . Then we set the key pair:  $(\mathbf{pk}, \mathbf{sk}, \mathbf{evk}) = (h, f, \gamma)$ .
- **YASHE.Enc( $m, \mathbf{pk}$ )**: To encrypt  $m \in \mathcal{R}_t$ , choose  $e, s \leftarrow \chi_{err}$  and then output the ciphertext  $\mathbf{ct} = [\lfloor \frac{q}{t} \rfloor \cdot [m]_t + e + hs]_q \in \mathcal{R}_q$ .
- **YASHE.Dec( $\mathbf{ct}, \mathbf{sk}$ )**: Given a ciphertext  $\mathbf{ct}$ , output  $\lfloor \frac{t}{q} \rfloor \cdot [f \cdot \mathbf{ct}]_q \pmod{t}$ . The inherent noise in the ciphertext is defined as the minimum value of infinite norm  $\|v\|_\infty = \max_i \{|v_i|\}$  such that  $f \cdot \mathbf{ct} = \lfloor \frac{t}{q} \rfloor \cdot [m]_t + v \pmod{q}$ .
- **YASHE.Add( $\mathbf{ct}, \mathbf{ct}'$ )/YASHE.Mult( $\mathbf{ct}, \mathbf{ct}', \mathbf{evk}$ )**: Given two ciphertexts  $\mathbf{ct}$  and  $\mathbf{ct}'$ , homomorphic addition is computed as  $\mathbf{ct}_{add} = [\mathbf{ct} + \mathbf{ct}']_q$ . Homomorphic multiplication is computed as  $\mathbf{ct}_{mult} = \text{SwitchKey}(\lfloor \frac{t}{q} \rfloor \mathbf{ct} \cdot \mathbf{ct}' \rfloor, \mathbf{evk})$  where the key switching function **SwitchKey** is used to trans-

## CHAPTER 2. PRELIMINARIES

form a ciphertext decryptable under the original secret key  $f$  (see [BLLN13] for details).

### 2.1.3 The Ring-GSW Scheme

Gentry, Sahai, and Waters [GSW13] suggested a fully homomorphic encryption based on the LWE problem, where the message is encrypted as an approximate eigenvalue of a ciphertext. Ducas and Micciancio [DM15] described its RLWE variant. The RGSW symmetric encryption scheme consists of the following algorithms.

- **RGSW.ParamsGen**( $\lambda$ ): Given the security parameter  $\lambda$ , choose  $m$  to be a power of two, modulus  $q$  with  $1 < q$ , discrete Gaussian distribution  $\chi_{err}$  of parameter  $\varsigma$ , and an integer base  $B_g$ . Output  $(m, q, \chi_{err}, B_g)$ .
- **RGSW.KeyGen**( $m, q, \chi_{err}, B_g$ ): On the input parameters, choose a polynomial  $\mathbf{s} \in \mathcal{R}_q$  which is chosen uniformly at random and set the secret key  $\mathbf{sk} = \mathbf{s}$ .
- **RGSW.Enc**( $m, \mathbf{sk}$ ): Let  $d_g = \lceil \log_{B_g}(q) \rceil$  and  $n = \phi(m) = m/2$ . To encrypt  $m \in \mathcal{R}_t$ , pick  $\mathbf{a} \in \mathcal{R}_Q^{2d_g}$  uniformly at random, and  $\mathbf{e} \in \mathcal{R}^{2d_g} \simeq \mathbb{Z}^{2d_g n}$  with discrete Gaussian distribution  $\chi$  of parameter  $\varsigma$ , and output the ciphertext

$$\mathbf{ct} = [\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + \mathbf{e}] + m\mathbf{G} \in \mathcal{R}_q^{2d_g \times 2}$$

where  $\mathbf{G} = (\mathbf{I}, B_g\mathbf{I}, \dots, B_g^{d_g-1}\mathbf{I}) \in \mathcal{R}_q^{2d_g \times 2}$ .

- **RGSW.Dec**( $\mathbf{ct}, \mathbf{sk}$ ): Let  $(c_0, c_1) \in \mathcal{R}_q^2$  be the first row vector of the given ciphertext  $\mathbf{ct}$ . Output  $c_0 - c_1 \cdot \mathbf{s} \pmod{q}$ .

Let  $\mathbf{WD}_{B_g}(\cdot)$  be the decomposition with the base  $B_g$ , where the dimension of input vector is multiplied by  $d_g$  through this algorithm. The RGSW encryption of  $m$  satisfies  $\mathbf{CT} \cdot (1, \mathbf{s}) = m \cdot (1, \mathbf{s}, \dots, B_g^{d_g-1}, B_g^{d_g-1}\mathbf{s}) + \mathbf{e}$ . Roughly,  $m$  is an *approximate* eigenvalue of  $\mathbf{WD}_{B_g}(\mathbf{CT})$  with respect to the eigenvector  $(1, \mathbf{s}, \dots, B_g^{d_g-1}, B_g^{d_g-1}\mathbf{s})$ .

## 2.2 Human Genome Comparison

Assume that there are two strings  $\alpha = \alpha_1 \dots \alpha_n$  and  $\beta = \beta_1 \dots \beta_m$  over an alphabet  $\Sigma$ . One can make another string with the same length by inserting spaces “–”, called *gaps*, and consider a matrix having two rows with these new strings. A gap in the first (resp. second) row is called **Insertion** (resp. **Deletion**). A column with the same (resp. distinct) characters is called **Match** (resp. **Mismatch**). Then the edit distance between two strings is the minimum number of these edit operations needed to transform one string into the other. Specifically, for two characters  $\alpha_i$  and  $\beta_j$ , let us define  $\text{sub}(\alpha_i, \beta_j)$  as follows:

$$\text{sub}(\alpha_i, \beta_j) = \begin{cases} 0 & \text{if } \alpha_i = \beta_j \text{ (Match),} \\ 1 & \text{if } \alpha_i \neq \beta_j \text{ (Mismatch).} \end{cases}$$

In Algorithm 1, we describe the *Wagner-Fischer edit distance algorithm* [WF74], and the edit distance is simply  $D(n, m)$ .

---

**Algorithm 1** Exact Edit Distance Algorithm

---

```

1: for  $i \leftarrow 0$  to  $n$  do
2:    $D_{i,0} \leftarrow i$ 
3: end for
4: for  $i \leftarrow 0$  to  $n$  do
5:   for  $j \leftarrow 0$  to  $m$  do
6:      $D_{0,j} \leftarrow j$ 
7:   end for
8: end for
9: for  $i \leftarrow 0$  to  $n$  do
10:  for  $j \leftarrow 0$  to  $m$  do
11:     $s \leftarrow (\alpha_i = \beta_j)? 0 : 1$ 
12:     $D(i, j) \leftarrow \min\{D(i-1, j-1) + s, D(i, j-1) + 1, D(i-1, j) + 1\}$ 
13:  end for
14: end for
15: return  $D(n, m)$ 

```

---

Tang et al. suggested an algorithm to compute the approximate edit distance between genome sequences. Suppose that two participants have Variation Call Format (VCF) files which contain genotype information such as

## CHAPTER 2. PRELIMINARIES

---

**Algorithm 2** Approximate Edit Distance Algorithm

---

```

1:  $e \leftarrow 0$ 
2: for  $i \in \mathcal{L}$  do
3:   if  $x_i == \emptyset$  then
4:      $D(x_i) \leftarrow 0$ 
5:   else if ' $x_i.sv$ ' == 'DEL' then
6:      $D(x_i) \leftarrow \text{len}(x_i.\text{ref})$ 
7:   else
8:      $D(x_i) \leftarrow \text{len}(x_i.\text{alt})$ 
9:   end if
10:  Define  $D(y_i)$  with the same way as  $D(x_i)$ 
11:  if  $((x_i.\text{ref} == y_i.\text{ref}) \text{ and } (x_i.\text{alt} == y_i.\text{alt}))$  then
12:     $e_i \leftarrow 0$ 
13:  else
14:     $e_i \leftarrow \max\{D(x_i), D(y_i)\}$ 
15:  end if
16:   $e \leftarrow e + e_i$ 
17: end for
18: return  $e$ 

```

---

chromosome number, position, reference and alternate sequences, where each base must be one of a single-nucleotide polymorphism (SNP): A, T, G, C. They also summarize some variants compared with reference genome (*e.g.*, insertion, deletion, or substitution). If there is only one record in VCF files at a specified location, the other one is considered to be an empty set ( $\emptyset$ ).

Let  $\mathcal{L}$  be a list indexed by the positions of two participants. Then we can define the approximate edit distance as described in Algorithm 2, where " $x_i.sv$ " denotes the type of structural variant relative to the reference, " $x_i.ref$ " the reference bases and " $x_i.alt$ " the alternate alleles. More precisely, given two data  $x_i$  and  $y_i$ , at the same locus from two samples, if  $x_i$  includes an insertion or deletion compared with the reference while  $y_i$  does not, use  $x_i$ 's distance as the edit distance at the current locus and then realign them to skip the inserted or deleted subsequences. If both of them have insertion or deletion at the same locus, choose the larger one as the approximation for the edit distance at the locus.

## Chapter 3

# Primitive Arithmetic Circuits under Homomorphic Encryption

We devise three primitives with bitwise encodings: an equality circuit, a comparison circuit and an integer addition circuit. We focus on a method for optimizing these circuits with respect to their depth and required homomorphic operations. For this purpose, we use SIMD along with automorphism operations. We also present the integer-based arithmetic circuits for equality and comparison.

**Notation.** All logarithms are base 2 unless otherwise indicated. By abuse of notation, we use “+” to denote homomorphic addition and  $\mathbf{HA}$  to denote the number of additions. Similarly, for homomorphic multiplication, we use “ $\cdot$ ” and  $\mathbf{HM}$ . When we say the (multiplicative) *depth*  $D(C)$  of a circuit  $C$  under homomorphic encryption, it means the total number of reduced levels in the circuit that is being evaluated homomorphically. Similarly  $\mathbf{HM}(C)$  denotes the number of homomorphic multiplications during evaluation.

## CHAPTER 3. PRIMITIVE ARITHMETIC CIRCUITS UNDER HOMOMORPHIC ENCRYPTION

### 3.1 Binary Arithmetic Circuits

When input messages are decomposed and encrypted in a bitwise manner, the encryption  $\bar{x}$  of a message  $x = x_\mu \cdots x_2 x_1$  means  $\{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_\mu\}$ , where  $x_i \in \{0, 1\}$ .

#### 3.1.1 Equality Circuit

For two  $\mu$ -bit integers  $x$  and  $y$ , we define an arithmetic circuit for the equality test as follows:

$$\mathbf{equal}(x, y) = \bigwedge_{i=1}^{\mu} (1 \oplus x_i \oplus y_i), \quad (3.1.1)$$

where we denote the XOR and AND gate by  $\oplus$  and  $\wedge$ . The output of  $\mathbf{equal}(\cdot, \cdot)$  is 1 in the case of equality and 0 otherwise. In the bit-sliced implementation, we assume that one ciphertext is used per bit; therefore, we have a total of  $2\mu$  ciphertexts to evaluate the equality test. If, rather than performing regular multiplication, we multiply each term after forming a binary-tree structure, the depth of the **equal** circuit becomes  $\log \mu$ . Specifically, the algorithm requires two homomorphic additions to compute  $1 + \bar{x}_i + \bar{y}_i$  and requires that  $\mu$  ciphertexts be multiplied by each other while consuming  $\log \mu$  depth.

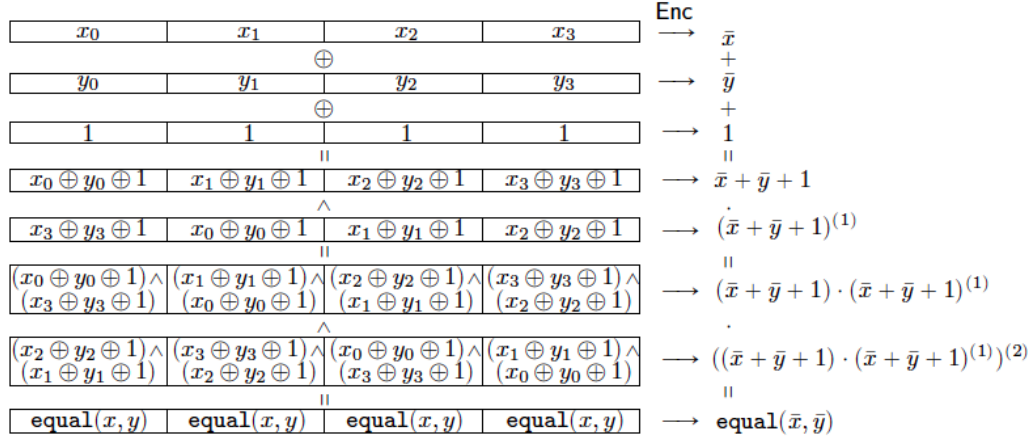
Our optimizations are focused on minimizing the number of homomorphic operations performed, particularly homomorphic multiplications. As shown by Smart and Vercauteren [SV14], we can pack each bit  $x_i$  into a single ciphertext. Next, we expand the right-hand side of Equation (3.1.1) and rearrange each term for convenience in SIMD execution. Then, we repeatedly apply SIMD operations to a vector of SIMD words. This procedure is the key to reduce the number of homomorphic multiplications from  $\mu - 1$  to  $\log \mu$ . We provide a better description of the complexity in Table 3.1.

For example, we consider the **equal** circuit for  $\mu = 4$ . We can pack each bit  $x_i$  and  $y_i$  into a corresponding single ciphertext. The application of the SIMD operations on this circuit is described in Fig. 3.1. The left side represents the execution over plaintext slots and the right side represents the execution



## CHAPTER 3. PRIMITIVE ARITHMETIC CIRCUITS UNDER HOMOMORPHIC ENCRYPTION

Figure 3.1: An Illustration of SIMD Execution for the `equal` Circuit



over the corresponding ciphertexts. We denote by  $\bar{z}^{(i)}$  the vector obtained by applying rotation by  $i$  to each element in  $\bar{z}$ . Namely, we implement a single automorphism  $X \mapsto X^{g^i}$  for some element  $g \in \mathbb{Z}_m^*$  of order  $\mu$  in the original group  $\mathbb{Z}_m^*$  and the quotient group  $\mathbb{Z}_m^*/\langle t \rangle$ . In Fig. 3.1, a rotation operation  $X \mapsto X^g$  is applied to the encryption  $\bar{x} + \bar{y} + 1$  for some element  $g \in \mathbb{Z}_m^*$  of order  $\mu = 4$ . Following the first SIMD homomorphic multiplication, we can obtain the encryption, which is entirely in the form of multiplications of two  $(x_i + y_i + 1)$ 's. In general, the parallel SIMD computation allows the equality circuit to be evaluated using an SWHE of depth  $\log \mu$  in  $\log \mu$  homomorphic multiplications.

### 3.1.2 Greater-Than Comparison Circuit

For two unsigned  $\mu$ -bit integers  $x$  and  $y$ , the circuit `com`( $x, y$ ) outputs 0 if  $x \geq y$  and 1 otherwise. This operation can be recursively defined as follows:

$$\text{com}(x, y) = c_\mu, \quad (3.1.2)$$

where  $c_i = (1 \oplus x_i) \wedge y_i + (1 \oplus x_i \oplus y_i) \wedge \bar{c}_{i-1}$  for  $i \geq 2$  with an initial value  $c_1 = (1 \oplus x_1) \wedge y_1$ .

## CHAPTER 3. PRIMITIVE ARITHMETIC CIRCUITS UNDER HOMOMORPHIC ENCRYPTION

As the first step of optimization, we express Equation (3.1.2) in the following closed form:

$$c_\mu = (1 \oplus x_\mu) \wedge y_\mu \oplus \bigoplus_{i=1}^{\mu-1} ((1 \oplus x_i) \wedge y_i \wedge d_{i+1} d_{i+2} \wedge d_\mu),$$

where  $d_k = (1 \oplus x_k \oplus y_k)$ . Because it has degree  $\mu + 1$ , we can deduce that the depth of the circuit is  $\log(\mu + 1)$ . Then it is easy to see that a naïve construction of the circuit incurs  $O(\mu^2)$  homomorphic multiplications.

The key observation is that the closed form is expressed as a sum of products of  $(1 + \bar{x}_i) \cdot \bar{y}_i$  and  $(1 + \bar{x}_i + \bar{y}_i)$  terms for  $i \in [1, \mu]$ . We are able to compute  $(1 + \bar{x}_i) \cdot \bar{y}_i$  for all  $i$  using only 1 homomorphic multiplication by applying the SIMD technique. Now, we must compute  $\prod_{k=i+1}^{\mu} \bar{d}_k$  for each  $i \in [1, \mu-1]$ . As stated above, a naïve method incurs  $O(\mu^2)$ ; however, using SIMD operations requires one to perform only  $2\mu - 4$  homomorphic multiplications, thereby consuming  $\log \mu$  depth. Finally, we need to multiply  $(1 + \bar{x}_i) \cdot \bar{y}_i$  by the result of the above computation, which also incurs only 1 homomorphic multiplication. Thus, the total number of homomorphic multiplications is equal to  $2\mu - 2$ .

**Remark 3.1.1.** We can address *signed* numbers by slightly modifying the circuit. Assume that we place a sign bit in the leftmost position of a value (*e.g.*, 0 for a positive number and 1 for a negative number) and use the two's-complement system. Then, for two  $\mu$ -bit values  $x$  and  $y$ ,  $\text{com}(\bar{x}, \bar{y}) = \bar{c}_\mu + \bar{x}_\mu + \bar{y}_\mu$ . It is evident that the case of two positive numbers corresponds to  $\bar{x}_\mu = \bar{y}_\mu = \bar{0}$ .

### 3.1.3 Integer Addition Circuit

Suppose that for two  $\mu$ -bit integers  $x$  and  $y$  and for an integer  $\nu > \mu$ , we construct two  $\nu$ -bit integers by padding with zeros on the left. Then a size- $\nu$  full adder  $\text{fadd}_\nu$  is recursively defined as follows:

$$\text{fadd}_\nu(x, y) = (s_1, s_2, \dots, s_\nu),$$

## CHAPTER 3. PRIMITIVE ARITHMETIC CIRCUITS UNDER HOMOMORPHIC ENCRYPTION

with the sums  $s_i = x_i \oplus y_i \oplus c_{i-1}$  and the carry-outs  $c_i = (x_i \wedge y_i) + \oplus((x_i \oplus y_i) \wedge c_{i-1})$  for  $i \in [2, \nu]$  with initial values  $s_1 = x_1 \oplus y_1$  and  $c_1 = x_1 \wedge y_1$ . The primary reason for considering such a large full adder is to cover SQL aggregate functions with many additions.

Our strategy for optimization is the same as above. Namely, we express each sum and carry in closed form and minimize the number of homomorphic operations using SIMD operations. Consequently, the  $\bar{s}_i$ 's are written as follows:  $\bar{s}_i = \bar{x}_i + \bar{y}_i + \sum_{j=1}^{i-1} t_{ij}$ , where  $t_{ij} = (\bar{x}_j \cdot \bar{y}_j) \prod_{j+1 \leq k \leq i-1} (\bar{x}_k + \bar{y}_k)$  for  $j < i - 1$  and  $t_{i,i-1} = \bar{x}_{i-1} \cdot \bar{y}_{i-1}$ . When  $i = \nu$  and  $j = 1$ , because  $\nu - 2$  homomorphic multiplications are required, the circuit has  $\log(\nu - 2)$  depth. However, we must perform an additional multiplication by  $\bar{x}_j \cdot \bar{y}_j$ . Thus, the total depth amounts to  $\log(\nu - 2) + 1$ . As before, the use of SIMD and parallelism via automorphism allows us to evaluate the integer addition circuit using only  $3\nu - 5$  homomorphic multiplications, whereas a naïve method requires  $\frac{(\nu^3 - 3\nu^2 + 8\nu)}{6}$  homomorphic multiplications.

Table 3.1: Complexity of circuit primitives

	Circuits	Complexity
Depth	<b>equal</b>	$\log \mu$
	<b>com</b>	$1 + \log \mu$
	<b>fadd</b>	$1 + \log(\nu - 2)$
Comp. <sup>†</sup>	<b>equal</b>	$2\text{HA} + (\log \mu) \text{HM}$
	<b>com</b>	$(\mu + 1 + \log \mu) \text{HA} + (2\mu - 2) \text{HM}$
	<b>fadd</b>	$\nu \text{HA} + (3\nu - 5) \text{HM}$

<sup>†</sup>Comp.: Computational complexity during homomorphic evaluations

### 3.1.4 Maximum & Minimum Circuits

In the following, we show that the comparison circuit leads to the the minimum circuits.

### CHAPTER 3. PRIMITIVE ARITHMETIC CIRCUITS UNDER HOMOMORPHIC ENCRYPTION

**Lemma 3.1.1.** *Given two  $\mu$ -bit values  $x = x_\mu \dots x_1$  and  $y = y_\mu \dots y_1$ , then  $z = z_\mu \dots z_1$  is the minimum value of  $x$  and  $y$  where*

$$z_i = (\text{com}(x, y) \wedge x_i) \oplus (1 \oplus \text{com}(x, y) \wedge y_i).$$

*Proof.* Let us denote a multiplication over integers by “ $\cdot$ ”. Then it is true that

$$\begin{aligned} \min\{x, y\} &= \text{com}(x, y) \cdot x + (1 \oplus \text{com}(x, y)) \cdot y \\ &= \text{com}(x, y) \cdot \left( \sum_{i=1}^{\mu} x_i \cdot 2^{i-1} \right) + (1 \oplus \text{com}(x, y)) \cdot \left( \sum_{i=1}^{\mu} y_i \cdot 2^{i-1} \right) \\ &= \sum_{i=1}^{\mu} \left( (\text{com}(x, y) \cdot x_i) + ((1 \oplus \text{com}(x, y)) \cdot y_i) \right) \cdot 2^{i-1}, \end{aligned}$$

where the inputs  $x$  and  $y$  can be written as binary representations in the second line. Since “ $\text{com}(x, y) \cdot x_i$ ” and “ $(1 \oplus \text{com}(x, y)) \cdot y_i$ ” cannot simultaneously be “1”, the lemma follows.  $\square$   $\square$

From Lemma 3.1.1, we define minimum circuits  $\text{min}^2 = (\text{min}_1^2, \dots, \text{min}_\mu^2)$  by

$$\text{min}_i^2 = (\text{com}(x, y) \wedge x_i) \oplus ((1 \oplus \text{com}(x, y)) \wedge y_i).$$

Then one can evaluate these circuits homomorphically with a SWHE scheme of depth  $(\log(\mu-1)+2)$ . We also obtain a natural generalization of computing the minimum value between many numbers: apply repeatedly the minimum circuits. Then this naive method has  $D(\text{min}^2) = (\log(\mu-1)+2) \cdot (\log k)$ .

On the other hand, we consider another way to compute the minimum value which requires circuits of lesser depth: Given  $\mu$ -bit values  $x^1, \dots, x^k$ ,

## CHAPTER 3. PRIMITIVE ARITHMETIC CIRCUITS UNDER HOMOMORPHIC ENCRYPTION

we define  $\min^k = (\min_1^k, \dots, \min_\mu^k)$  by  $\min_i^k = \bigoplus_{j=1}^k (\mathbf{c}_j \wedge x_i^j)$  where

$$\mathbf{c}_j = \begin{cases} \mathbf{com}(x^1, x^2) \wedge \dots \wedge \mathbf{com}(x^1, x^k) & \text{if } j = 1, \\ (1 \oplus \mathbf{com}(x^1, x^j)) \wedge \dots \wedge (1 \oplus \mathbf{com}(x^{j-1}, x^j)) \wedge \\ \quad \mathbf{com}(x^j, x^{j+1}) \wedge \dots \wedge \mathbf{com}(x^j, x^k) & \text{if } 2 \leq j \leq k-1, \\ (1 \oplus \mathbf{com}(x^1, x^k)) \wedge \dots \wedge (1 \oplus \mathbf{com}(x^{k-1}, x^k)) & \text{if } j = k. \end{cases}$$

It is easy to show that this method has

$$\begin{aligned} D(\min^k) &= \log(\mu - 1) + \log(k - 1) + 2, \\ \text{HM}(\min^k) &= \left( 2\mu - 3 + \frac{(\mu - 1) \log(\mu - 1)}{2} \right) \frac{(k - 1)(k - 2)}{2} + k(k - 2 + \mu). \end{aligned}$$

More specifically, it first needs  $\log(\mu - 1) + 1$  levels to compute  $\mathbf{com}(x^i, x^j)$ 's by Lemma 1. Then it requires  $\log(k - 1)$  levels to compute  $\mathbf{c}_j$  for  $1 \leq j \leq k$  by multiplying  $\mathbf{com}(x^i, x^j)$ 's with a binary tree. Finally, the claim follows because we need one more level to compute  $\mathbf{c}_j \wedge x_i^j$ .

### 3.2 Arithmetic Circuits over the Integers

We now present the integer-based arithmetic circuits for equality and comparison. We first describe the basic concept that underlies our modifications of the binary arithmetic circuits as described above. We also represent the arithmetic circuits over the integers by using Fermat's Little Theorem and Lagrange Interpolation Formula.

Note that if the plaintext modulus is sufficiently large, we are able to perform addition without overflow in the plaintext space. This means addition can be securely performed with degree 1 circuit.

## CHAPTER 3. PRIMITIVE ARITHMETIC CIRCUITS UNDER HOMOMORPHIC ENCRYPTION

### 3.2.1 Calibrating Binary Circuit Primitives

It is well known that for  $x, y \in \{0, 1\}$ , the following properties hold:

$$x \oplus y = x + y - 2 \cdot x \cdot y \quad \text{and} \quad x \wedge y = x \cdot y,$$

where  $+$ ,  $-$ , and  $\cdot$  are arithmetic operations over the integers. Based on this observation, our equality test can be rewritten as follows:

$$\text{equal}(x, y) = \prod_{i=1}^{\mu} (1 - x_i - y_i + 2 \cdot x_i \cdot y_i) = \prod_{i=1}^{\mu} (1 - (x_i - y_i)^2).$$

For only a small additional cost, we can construct a new arithmetic circuit for an equality test operating under homomorphic encryption. Next, consider the `com` circuit over the integers. Recall that the closed form of  $c_\mu$  is

$$c_\mu = (1 - x_\mu) \cdot y_\mu + \sum_{i=1}^{\mu-1} (1 - x_i) \cdot y_i \cdot (d_{i+1} d_{i+2} \cdots d_\mu).$$

Rather than  $d_j = (1 + x_j + y_j)$ , we set

$$d_j = (1 + 2 \cdot x_j \cdot y_j - y_j - x_j) \cdot (1 + 2 \cdot x_j \cdot y_j - 2y_j).$$

However, the number of ciphertexts is still too large since it requires bit-wise encodings for equality and comparison and each bit is considered as an element of large message space.

### 3.2.2 Arithmetic Circuits over the Integers Based on Fermat's Little Theorem

It follows from Fermat's Little Theorem that if  $x \in \mathbb{Z}$  and  $p$  is a prime not dividing  $x$ , then  $p$  divides  $x^{p-1} - 1$ , that is,  $x^{p-1} \equiv 1 \pmod{p}$ . Then we can

## CHAPTER 3. PRIMITIVE ARITHMETIC CIRCUITS UNDER HOMOMORPHIC ENCRYPTION

define the *indicator* function of the integer  $k$  as follows:

$$\mathbf{1}_k(x) = \begin{cases} 1 & \text{if } x = k \\ 0 & \text{o.w.} \end{cases} \equiv 1 - (x - k)^{p-1} \pmod{p},$$

which implies that  $\mathbf{equal}(x, y) \equiv \mathbf{1}_0(x - y) \pmod{p}$ .

Similary, we consider the indicator function of the set  $[0, \frac{p-1}{2}] \cap \mathbb{Z}$  as follows:

$$\mathbf{1}_{[0, \frac{p-1}{2}]}(x) = \begin{cases} 1 & \text{if } 0 \leq x \leq \frac{p-1}{2}, \\ 0 & \text{if } \frac{p-1}{2} < x \leq p-1. \end{cases}$$

This function is written as the sum of indicator functions from 0 to  $\frac{p-1}{2}$ , that is,  $\mathbf{1}_{[0, \frac{p-1}{2}]}(x) \equiv \sum_{k=0}^{\frac{p-1}{2}} \mathbf{1}_k(x) \pmod{p}$ . Thus the `com` circuit is represented as  $\mathbf{com}(x, y) \equiv \mathbf{1}_{[0, \frac{p-1}{2}]}([y - x]_p) \pmod{p}$ . Even though there are not so many ciphertexts compared with the method in Section 3.2.1, they require large depth as about  $\log p$  and high complexity as  $\log p$  and  $O(p \log p)$  for the equality and greater-than comparison circuits, respectively.

### 3.2.3 Arithmetic Circuits over the Integers Based on Lagrange Interpolation Formula

If  $x \in [0, \ell] \cup [p - \ell, p - 1]$  and  $k \in [0, \ell]$ , Lagrange Interpolation formula gives

$$\begin{aligned} \mathbf{1}_k(x) &= \begin{cases} 1 & \text{if } x = k, \\ 0 & \text{if } [1, \ell] \cup [p - \ell, p - 1], \end{cases} \\ &\equiv M_k \cdot \prod_{i=0, i \neq k}^{\ell} (x - i) \cdot \prod_{i=p-\ell}^{p-1} (x - i) \pmod{p}. \end{aligned}$$

### CHAPTER 3. PRIMITIVE ARITHMETIC CIRCUITS UNDER HOMOMORPHIC ENCRYPTION

If precomputing  $M_k$  as a constant value, the total depth is  $\log \ell + 2$ . We can easily extend the indicator function to the set  $[0, \ell] \cap \mathbb{Z}$  as follows:

$$\begin{aligned} \mathbf{1}_{[0, \ell]}(x) &= \begin{cases} 1 & \text{if } x \in [0, \ell] \\ 0 & \text{if } x \in [p - \ell, p - 1] \end{cases} \\ &\equiv \sum_{k=0}^{\ell} \mathbf{1}_k(x) \pmod{p}. \\ &\equiv \prod_{i=p-\ell}^{p-1} (x - i) \cdot \left( \sum_{k=0}^{\ell} (M_k \cdot \prod_{i=0, i \neq k}^{\ell} (x - i)) \right) \pmod{p}. \end{aligned}$$

If precomputing  $M_k$ 's as constant values, the total depth can be reduced to about  $\log 2\ell$ .

Now suppose that the difference between  $x$  and  $y$  is less than  $\ell$ . Then we can derive the arithmetic circuits using the defined indicator functions as follows:

- $\text{equal}(x, y) = \mathbf{1}_0(x - y)$
- $\text{com}(x, y) = \mathbf{1}_{[0, \ell]}([y - x]_p)$
- $\text{min}(x, y) = \text{com}(x, y) \cdot (x - y) + y$

Therefore, a SWHE scheme of depth  $\log \ell$  can evaluate the equality circuit (resp. greater-than comparison circuit) in  $2\ell$  (resp.  $\ell^2$ ) homomorphic computations.



## Chapter 4

# Private Database Query Processing

In this section, we demonstrate how to efficiently perform queries on encrypted data using the proposed circuit primitives. We first describe our techniques in a general setting and then show how our ideas apply to database applications.

### 4.1 General-Purpose Search-and-Compute

We begin by describing our basic idea for performing a *search* operation over encrypted data. We assume that a collection of data is partitioned into  $N$   $\mu$ -bit items denoted by  $R_1 \parallel \cdots \parallel R_N$  and that the data have been encrypted and stored in the form of  $\bar{R}_1 \parallel \cdots \parallel \bar{R}_N$ .

For a predicate  $\varphi$  on a ciphertext  $\mathcal{C}$ , a search on encrypted data outputs  $\bar{R}_i$  if  $\varphi(\bar{R}_i) = \bar{1}$  and  $\bar{0}$  otherwise. More formally, let  $\varphi : \mathcal{C} \rightarrow \{\bar{0}, \bar{1}\}$  be a predicate on encrypted data. Then, we say that  $S_\varphi : \mathcal{C}^N \rightarrow \mathcal{C}^N$  is a search on the encrypted data and is defined as follows:

$$S_\varphi(\bar{R}_1, \dots, \bar{R}_N) := (\varphi(\bar{R}_1) \cdot \bar{R}_1, \dots, \varphi(\bar{R}_N) \cdot \bar{R}_N).$$

We then extend this operation to a more general operation on encrypted

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

data, *i.e.*, a search-and-compute operation on encrypted data, as follows. Let  $F : \mathcal{C}^N \rightarrow \mathcal{C}$  be an arithmetic function on encryptions. Then, for a restricted search  $S_\varphi : \mathcal{C}^N \rightarrow \mathcal{C}^N$ , we say that  $(F \circ S_\varphi)(\bar{R}_1, \dots, \bar{R}_N)$  is a search-and-compute operation on encryptions.

Furthermore, we quantify the efficiency of such search-and-compute operations on encrypted data in Theorem 4.1.1. This theorem states that if we can perform a search on encrypted data restricted by a  $\varphi$  that specifies only the equality operator, then a search query on the encrypted data requires  $N(2\text{HA} + \log \mu\text{HM})$  homomorphic operations in total. If the predicate  $\varphi$  allows one to specify all the comparison operators in the set  $\{<, \leq, >, \geq, \neq\}$ , then we can perform  $S_\varphi(\bar{R}_1, \dots, \bar{R}_N)$  using  $O(\mu N)$  homomorphic multiplications.

**Theorem 4.1.1.** *Let  $\text{HM}(\varphi)$  and  $\text{HM}(F)$  be the total numbers of homomorphic multiplications for  $\varphi$  and  $F$ , respectively. Then, we can perform  $(F \circ S_\varphi)(\bar{R}_1, \dots, \bar{R}_N)$  using  $O(N(\text{HM}(\varphi)) + \text{HM}(F))$  homomorphic operations. Specifically, we can perform a search on encrypted data restricted by  $\varphi$  using at most  $O(N(\text{HM}(\varphi)))$  homomorphic operations.*

*Proof.* Because homomorphic multiplication dominates the performance of the operation, we may consider only operations of this type. Because the predicate  $\varphi$  requires  $O(\text{HM}(\varphi))$  homomorphic operations, we see that  $S_\varphi$  requires  $O(N(\text{HM}(\varphi)))$  homomorphic operations to compute the predicate  $N$  times. Thus, the operation uses  $O(\text{HM}(F))$  homomorphic operations to evaluate an arithmetic function  $F$  on encrypted data. Therefore, we can conclude that the total computation complexity of the search-and-compute operation on encryptions is  $O(N(\text{HM}(\varphi)) + \text{HM}(F))$ . In particular, when we consider a search on encrypted data,  $F$  can be regarded as the identity map. Therefore, we can perform a search on encrypted data restricted by  $\varphi$  using at most  $O(N(\text{HM}(\varphi)))$  homomorphic operations.  $\square$

### 4.1.1 A High-level Overview of Our Approach

Figure 4.1 graphically illustrates the high-level architecture of our approach. Assuming a database consisting of  $N$  blocks, *i.e.*,  $R_1 \parallel R_2 \parallel \dots \parallel R_N$ , to encrypt the record  $R_i$ , a DB user prepares a pair of public/private keys  $(\mathbf{pk}, \mathbf{sk})$  for an FHE scheme and publishes the public key to a DB server. The DB users store their encrypted records  $\bar{R}_i = \text{Enc}(R_i, \mathbf{pk})$  for  $1 \leq i \leq N$  in the same way as normal write queries (*e.g.*, using the `insert-into` statement). We use an efficient variant of an FHE scheme: a somewhat homomorphic encryption scheme.

Suppose that the user wants to submit a retrieval query  $Q$  to the DB server. Before being submitted, the query  $Q$  needs to be properly pre-processed so that all clear messages, such as constant values, are encrypted under the public key  $\mathbf{pk}$ . We denote this transformed query by  $\bar{Q}$ .

Upon receiving  $\bar{Q}$ , the DB server compiles it into  $\bar{Q}^*$  by applying our techniques. The readers can consider a dedicated module for performing this task.\* Hereafter, we call the module a *Private Search-and-compute* (PSnC) processor. Next, the DB server homomorphically evaluates  $\bar{Q}^*$  over the fully encrypted databases and returns the resulting ciphertexts to the user.

The DB user can decrypt the output using his private key  $\mathbf{sk}$  while learning no additional data except for the records satisfying the `where` conditions.

### 4.1.2 Security Evaluation

Secrecy against a semi-honest DB server is ensured because encrypted data cannot be leaked due to the semantic security of our underlying SWHE scheme. Secrecy against a semi-honest DB user therefore follows because the result of a query expressed by our circuit primitives is equivalent to  $\bar{0}$  if the specified conditions do not hold; therefore, the resulting ciphertext is equal to  $\bar{0}$ . This implies that the evaluated ciphertexts do not leak any information

---

\*Alternatively, one may imagine that  $\bar{Q}^*$  transformed by the DB user directly is sent to the DB server. However, considering optimization and performance, we believe that the better choice involves the module becoming part of the DBMS.

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

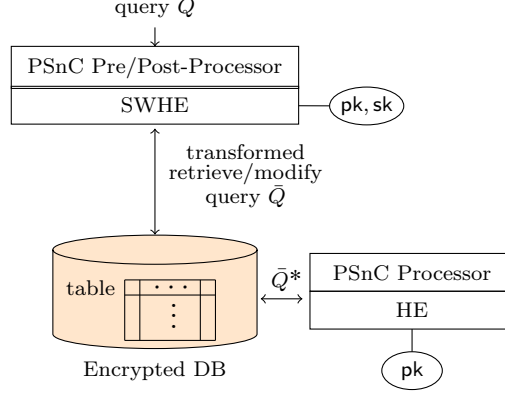


Figure 4.1: Our PSnC Framework

except for the number of unsatisfied tuples.

## 4.2 Applications to Encrypted Databases

We use  $R(A_1, \dots, A_d)$  to denote a relation schema  $R$  of degree  $d$  consisting of attributes  $A_1, \dots, A_d$ , and we use  $\bar{A}_j$  to denote the corresponding encrypted attribute. As mentioned above, we use  $A_j^{(i)}$  to denote the  $j$ -th attribute value of the  $i$ -th tuple, and for convenience, we assume that each has a length of  $\mu$  bits.

### 4.2.1 Search Queries

**Simple Selection Queries.** Consider a simple retrieval query, as follows:

$$\text{select } A_{j_1}, \dots, A_{j_s} \text{ from } R \text{ where } A_{j_0} = \alpha; \quad (\text{Q}.1)$$

where  $\alpha$  is a constant value.

An efficient construction of (Q.1) using our `equal` circuit is as follows:

$$\text{equal} \left( \bar{A}_{j_0}^{(i)}, \bar{\alpha} \right) \cdot \left( \bar{A}_{j_1}^{(i)}, \dots, \bar{A}_{j_s}^{(i)} \right) \quad (\bar{\text{Q}}^*.1)$$

for each  $i \in [1, N]$ . It follows from Theorem 4.1.1 that  $(\bar{\text{Q}}^*.1)$  has the com-

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

Table 4.1: Complexity of Search Queries

	Query	Complexity
Depth	$(\bar{Q}^*.1)$	$1 + \log \mu$
	$(\bar{Q}^*.2)$	$1 + \log \mu + \log \tau$
Comp.	$(\bar{Q}^*.1)$	$2NHA + N(1 + \log \mu) \text{ HM}$
	$(\bar{Q}^*.2)$	$2\tau NHA + \tau N(1 + \log \mu) \text{ HM}$

plexity evaluation given in Table 4.1.

**Conjunctive & Disjunctive Queries.** The query (Q.1) can be extended by adding one or more conjunctive or disjunctive conditions to the **where** clause. Consider a conjunctive query as follows:

$$\begin{aligned}
 &\textbf{select} \quad A_{j_1}, \dots, A_{j_s} \\
 &\quad \textbf{from} \quad R \\
 &\quad \textbf{where} \quad A_{j'_1} = \alpha_1 \textbf{ and } \dots \textbf{ and } A_{j'_\tau} = \alpha_\tau;
 \end{aligned} \tag{Q.2}$$

The query (Q.2) is expressed as follows: For each  $i \in [1, N]$ ,

$$\prod_{k=1}^{\tau} \textbf{equal} \left( \bar{A}_{j'_k}^{(i)}, \bar{\alpha}_k \right) \cdot \left( \bar{A}_{j_1}^{(i)}, \dots, \bar{A}_{j_s}^{(i)} \right). \tag{\bar{Q}^*.2}$$

A disjunctive query whose logical connectives are all **ors** can also be evaluated by changing the predicate to

$$\left( 1 + \prod_{k=1}^{\tau} \left( \textbf{equal} \left( \bar{A}_{j'_k}^{(i)}, \bar{\alpha}_k \right) + 1 \right) \right).$$

With  $\tau$  denoting the number of connectives,  $(\bar{Q}^*.2)$  requires an additional depth of  $\log \tau$  compared with  $(\bar{Q}^*.1)$  to compute the multiplications among the  $\tau$  equality tests. Table 4.1 reports the complexity analysis.

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

### 4.2.2 Search-and-Compute Queries

We continue to present important real constructions as an extension of Theorem 4.1.1, in which  $F$  is one of the built-in SQL aggregate functions: `sum`, `avg`, `count` and `max`. We begin with the case of  $F = \text{sum}$ .

**Search-and-sum Query.** Consider the following `sum` query:

$$\text{select sum}(A_{j_1}) \text{ from } R \text{ where } A_{j_0} = \alpha; \quad (\text{Q}.3)$$

As mentioned above, because our plaintext space is  $\mathbb{Z}_2$ , repeatedly applying simple homomorphic additions does not ensure correctness, which is the motivation for our integer addition circuit (see Section 3.1.3). Using this circuit, we can efficiently perform (Q.3), which is expressed as follows:

$$\text{fadd}_{\mu + \log N} \left( \text{equal} \left( \bar{A}_{j_0}^{(i)}, \bar{\alpha} \right) \cdot \bar{A}_{j_1}^{(i)} \right). \quad (\bar{\text{Q}}^*.3)$$

Because the result of the search-and-sum query is less than  $2^\mu N$ , using a full adder of size  $\nu = \mu + \log N$  to add all the values is sufficient. Using our optimized equality circuit,  $(\bar{\text{Q}}^*.3)$  requires  $N$  equality tests in total and  $N$  homomorphic multiplications for each result of the test. Thus, the total computational cost is  $(2N + \nu(N - 1))\text{HA} + (N(1 + \log \mu) + (N - 1)(3\nu - 5))\text{HM}$  with the depth  $1 + \log \mu + \log N(1 + \log(\nu - 2))$  according to Theorem 4.2.1 below.

**Theorem 4.2.1.** *Let  $|R|$  denote the cardinality of a set of tuples from a relation schema  $R$ . Suppose that all keyword attributes in the `where` clause and all numerical attributes in the `select` clause have  $\|kwd\|$  bits and  $\|num\|$  bits, respectively. Then, a search-and-sum query can be processed with the depth  $1 + \lceil \log(\|kwd\|) \rceil + \lceil \log |R| \rceil \cdot (1 + \lceil \log(\|num\| + \lceil \log |R| \rceil - 2) \rceil)$ .*

*Proof.* The query  $(\bar{\text{Q}}^*.3)$  consumes  $1 + \lceil \log(\|kwd\|) \rceil$  levels for the computation of all equality tests. Then, it performs  $(|R| - 1)$  full-adder operations

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

on the results, each of which is of size  $(\|num\| + \lceil \log |R| \rceil)$  and consumes  $(1 + \lceil \log (\|num\| + \lceil \log |R| \rceil - 2) \rceil)$  levels.  $\square$

**Search-and-Count Query.** We observe that search-and-count queries can be processed in a similar manner. For example, assume a search-and-count query with `count(*)` in place of `sum( $A_{j_1}$ )` in (Q.3). This query can also be efficiently processed using `faddlog N (equal ( $\bar{A}_{j_0}^{(i)}, \bar{\alpha}$ ))`.

**Search-and-Avg Query.** To process a search-and-compute query with the `avg` aggregate function, it is sufficient to compute multiple search-and-sum queries because an average can then be obtained by applying one division after decryption.

**Search-and-Max(Min) Query.** It is evident that one can obtain the `max` (or `min`) aggregate function by repeatedly applying the `com` circuit primitive.

### 4.2.3 Join Queries

Now, we design join queries within the search-and-compute paradigm. Suppose that we have another relation  $S(B_1, \dots, B_e)$  consisting of  $M$  tuples, where  $M \leq N$ . First, we consider a simple join query, as follows:

$$\begin{aligned} &\text{select} \quad r.A_{j_1}, \dots, r.A_{j_s}, s.B_{j'_1}, \dots, s.B_{j'_{s'}} \\ &\quad \text{from} \quad R \text{ as } r, S \text{ as } s \\ &\quad \text{where} \quad r.A_{j_k} = s.B_{j'_{k'}}; \end{aligned} \tag{Q.4}$$

Then, this type of query can be expressed as follows: For each  $i \in [1, N], i' \in [1, M]$ ,

$$\text{equal} \left( r.\bar{A}_{j_k}^{(i)}, s.\bar{B}_{j'_{k'}}^{(i')} \right) \cdot \left( r.\bar{A}_{j_1}^{(i)}, s.\bar{B}_{j'_1}^{(i')}, \dots \right). \tag{Q*.4}$$

For fixed  $i$  and  $i'$ , we suppose that each numeric-type attribute is packed in only one ciphertext. Then, the only difference from (Q\*.1) is that (Q\*.4)

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

requires two homomorphic multiplications by the results of the search operations; thus, we must perform  $NM$  equality tests in total. Hence, the depth of the circuit needed to process  $(\bar{Q}^*.4)$  is  $1 + \log \mu$ , and the computational complexity is  $(2NM) \text{ HA} + NM(2 + \log \mu) \text{ HM}$ .

Next, we consider an advanced join query  $(Q.5)$  with two aggregate functions  $\text{sum}(r.A_j)$  and  $\text{count}(*)$  and the same simple condition as for  $(Q.4)$ . Assuming  $\text{sum}(r.A_j) < 2^\mu NM$ , we use a full adder of size  $\nu = \mu + \log(NM)$ . By contrast, the result of  $\text{count}(*)$  is  $< NM$ , and it is sufficient to use a full adder of size  $\log(NM)$ . Thus, one candidate circuit construction for  $(Q.5)$  is as follows:

$$\begin{aligned} & \text{fadd}_{\mu + \log NM} \left( \text{equal} \left( r.\bar{A}_{j_k}^{(i)}, s.\bar{B}_{j_{k'}}^{(i')} \right) \cdot r.\bar{A}_j^{(i)} \right), \\ & \text{fadd}_{\log NM} \left( \text{equal} \left( r.\bar{A}_{j_k}^{(i)}, s.\bar{B}_{j_{k'}}^{(i')} \right) \right). \end{aligned} \quad (\bar{Q}^*.5)$$

With respect to  $\text{sum}(r.A_j)$ , this is identical to  $(\bar{Q}^*.3)$  except for the number of operands of the additions. Therefore, the depth for evaluation amounts to  $1 + \log \mu + \log(NM)(1 + \log(\nu - 2))$ , and the computation complexity is  $(2NM + \nu(NM - 1))\text{HA} + (NM(1 + \log \mu) + (NM - 1)(3\nu - 5))\text{HM}$ .

### 4.3 Performance Improvement

There is still room to further improve the performance of the circuit primitives in Section 3. Our strategies consist of three interrelated components: switching the message space  $\mathbb{Z}_2$  to  $\mathbb{Z}_t$ ; adapting the circuit primitives to  $\mathbb{Z}_t$ ; and fine-tuning the circuit primitives, again using SIMD operations.

#### 4.3.1 Larger Message Spaces with Lazy Carry Processing

If we encrypt messages in a bit-by-bit manner, the primary advantage is that the two comparison operations are very cheap; however, applying an integer addition circuit to encrypted data is expensive (see Table 4.2). Instead, it



## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

would be of substantial benefit to take the message domain to be a large integer ring if doing so would allow one to efficiently evaluate the addition circuit with much lesser depth. One of the important motivations for using such a large message space is that the bit length of the keyword attributes (*e.g.*,  $\leq 20$  bits) in the **where** clause is generally smaller than that of the numeric-type attributes (*e.g.*,  $\geq 30$  bits) in the **select** clause.

Specifically, if we represent a numeric-type attribute  $A$  in the radix  $2^\omega$ , then we have

$$\sum_i A^{(i)} = \sum_k \sum_i [A^{(i)}]_k \cdot (2^\omega)^k;$$

therefore, it is sufficient to compute  $\sum_i [A^{(i)}]_k$  over the integers. Assuming that the plaintext modulus  $t$  is sufficiently large, we are able to perform addition without overflow in  $\mathbb{Z}_t$ . Note that we only need to process carry operations after computing each of them over the large integer ring.

To verify the performance improvement achieved through integer encoding, we report the running time for each circuit primitive in Table 4.2. We suspect that integer encoding yields greater benefits in the performance of search-and-compute queries because aggregate functions rely extensively on addition. The experimental results presented in Table 4.2 used  $10^2$  integers in  $\mathbb{Z}_t$  randomly generated by the NTL library routines.

Table 4.2: Running-time comparisons in  $\mathbb{Z}_2$  and  $\mathbb{Z}_{2^{14}}$

Msg Space	equal (10 bits)	com (10 bits)	add (30 bits)
$\mathbb{Z}_2$	2.2621 ms	8.5906 ms	228.5180 ms
$\mathbb{Z}_{2^{14}}$	208.6543 ms	307.5200 ms	0.0004 ms

### 4.3.2 Calibrating Circuit Primitives

It is clear that the use of a different message space must result in modifications to our circuit primitives. Prior to discussing our modifications in

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

detail, we must determine certain lower bounds on the depth for homomorphic multiplication as a function of  $t$ . We have two types of homomorphic multiplications: multiplying a ciphertext by another ciphertext and multiplying a ciphertext by a known constant. We formally state the corresponding depth bounds in Theorem 4.3.1.

**Theorem 4.3.1.** *Suppose that the native message space of the BGV cryptosystem is a polynomial ring  $\mathbb{Z}_t[X]/\langle\Phi_m(X)\rangle$  and that a chain of moduli is defined by a set of primes of approximately the same size,  $p_0, \dots, p_L$ , that is, the  $i$ -th modulus  $q_i$  is defined as  $q_i = \prod_{k=0}^i p_k$ . For simplicity, assume that  $p$  is the size of the  $p_k$ s. Let us denote by  $h$  the Hamming weight of the secret key. For  $i \leq j$ , let  $\mathcal{C}$  and  $\mathcal{C}'$  be normal ciphertexts at levels  $i$  and  $j$ , respectively. Then, the depth for the multiplication of  $\mathcal{C}$  and  $\mathcal{C}'$ , which is denoted by  $\tilde{d}$ , is the smallest nonnegative integer that satisfies the following inequality:  $t^2 \cdot \phi(m) \cdot (1+h) \cdot ([q_i^{-1}]_t)^2 < 6p^{2\tilde{d}}$ . In addition, the depth for the multiplication of  $\mathbf{c}$  by a constant, which is denoted by  $\tilde{d}_c$ , is the smallest nonnegative integer for which the following inequality holds:  $\phi(m) \cdot (t/2)^2 < p^{2\tilde{d}_c}$ .*

*Proof.* Before multiplying two ciphertexts, we set their noise magnitude to be smaller than the pre-set constant  $B = t^2\phi(m)(1+h)/12$  via modulus switching. Subsequently, we obtain a tensor product of the ciphertexts, and the result has a noise magnitude of  $2B([q_i^{-1}]_t)^2$ . Next, scale-down is performed by removing small primes  $p_k$  from the current prime set of the tensored ciphertext; we use  $\Delta$  to denote the product of the removed primes. We then have  $2B^2([q_i^{-1}]_t)^2/\Delta^2 < B$ . By assumption, it may be considered that  $\Delta = p^{\tilde{d}}$ , which means that  $\tilde{d}$  is the smallest nonnegative integer that satisfies the inequality  $2B([q_i^{-1}]_t)^2 < p^{2\tilde{d}}$ .

We now consider the case in which  $\mathbf{c}$  is multiplied by a constant. As above, we obtain a noise estimate of  $B \cdot \phi(m) \cdot (t/2)^2$ . Thus, we see that  $\tilde{d}_c$  is the smallest nonnegative integer that satisfies the inequality  $\phi(m) \cdot (t/2)^2 < p^{2\tilde{d}_c}$ .  $\square$

Table 4.3 presents the complexity results for search-and-compute queries

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

on encrypted databases of  $N$  tuples with  $\mu$ -bit attributes that are obtained when using the new message space  $\mathbb{Z}_t$ .

Table 4.3: Complexity of search-and-sum queries

	Query type in <b>where</b> clause	Complexity
Depth	<b>equal</b>	$(2 + \log \mu) \tilde{d} + \tilde{d}_c$
	<b>conj<math>_{\tau}</math></b>	$(2 + \log \mu + \log \tau) \tilde{d} + \tilde{d}_c$
	<b>com</b>	$(4 + \log \mu) \tilde{d} + \tilde{d}_c$
Comp.	<b>equal</b>	$(4N - 1) \text{HA} + N(3 + \log \mu) \text{HM}$
	<b>conj<math>_{\tau}</math></b>	$((3\tau + 1)N - 1) \text{HA} + \tau N(3 + \log \mu) \text{HM}$
	<b>com</b>	$(N(\mu + 5 + \log \mu) - 1) \text{HA} + N(2\mu + 1) \text{HM}$

### 4.4 Implementation and Discussion

This section demonstrates the performance of query processing expressed by our optimized circuit primitives. The essential goal of the experiments in this section is to verify the efficiency of our solution in terms of performance. Thus, we reported the experimental results for each query. We performed a somewhat fair comparison with the prior related works in [LNV11, BGH<sup>+</sup>13], although each work is fairly different from its underlying SWHE scheme and experimental settings.

All experiments reported in our paper were performed on a machine with an Intel Xeon 2.3 GHz processor with 192 GB of main memory running a Linux 3.2.0 operating system. All methods were implemented using the GCC compiler version 4.2.1. In our experiments, we used a variant of a BGV-type SWHE scheme [GHS12] with Shoup’s NTL library [S<sup>+</sup>01] and Shoup-Halevi’s HE library [HS13]. Throughout this section, when we measured the average running times, we excluded computing times used in data encryption and decryption.

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

### 4.4.1 Adjusting the Parameters

Without a loss of generality, we assume that the bit length of keyword attributes in the **where** clause is 10-bit and that of numeric-type attributes in the **select** clause is 30-bit. The keyword attributes are expressed in a bit-by-bit manner, and each bit is an element of  $\mathbb{Z}_{2^r}$ . In addition, numeric-type attributes are expressed by the radix  $2^\omega$  but are still in the same space  $\mathbb{Z}_{2^r}$ .

We begin by observing the following relation among the parameters. At this point, we consider the *selectivity* of a selection condition, which means the fraction of tuples that satisfies the condition, and we denote it by  $\varepsilon$ .

**Theorem 4.4.1.** *Let  $A$  be a numeric-type attribute. For a positive integer  $\omega \geq 1$ , suppose that each attribute is written as  $A = \sum_k [A]_k \cdot (2^\omega)^k$  with  $0 \leq [A]_k < 2^\omega$ . Then, to process a search-and-sum query, one can take a plaintext modulus with  $r = \Theta(\omega + \log(\varepsilon \cdot N))$ . Similarly, for a search-and-count query, it suffices to choose the parameter  $r$  so that  $r = \Theta(\log(\varepsilon \cdot N))$ .*

*Proof.* The goal of the theorem is to provide a bound for the size of a plaintext modulus; therefore, we simply omit an overhead bar for all variables. Let us denote by  $\varphi$  a predicate on encrypted data and by  $A^*$  a keyword attribute. Then, a search-and-sum query can be written as

$$\sum_i S_\varphi(A^*, \alpha) \cdot A^{(i)} = \sum_k \left( \sum_i S_\varphi(A^*, \alpha) \cdot [A^{(i)}]_k \right) \cdot (2^\omega)^k.$$

We then have that

$$\sum_i S_\varphi(A^*, \alpha) \cdot [A^{(i)}]_k < 2^\omega \sum_i S_\varphi(A^*, \alpha) = 2^\omega \cdot (\varepsilon N).$$

Thus, for a database with  $N$  records, it is sufficient to choose  $r$  such that  $2^\omega \cdot (\varepsilon N) \leq 2^r$ . Note, the larger we make the plaintext modulus  $2^r$ , the more noise there is in the ciphertexts and thus the faster we consume the ciphertext level. Therefore, it appears that  $\omega + \log(\varepsilon N)$  is the tight bound for the parameter  $r$ .

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

Because a search-and-count query does not need to consider a specific attribute, we immediately know that  $\sum_i S_\varphi(A^*, \alpha) = \varepsilon N < 2^r$ .  $\square$

One may wonder why  $S_\varphi(\cdot, \dots)$  does not take multiple keyword attributes in the proof. Because we consider the selectivity ratio, it does not need to do so. In our experiments, we varied the selectivity ratio from 5 to 40% and plotted the average running time of queries over a database with  $N = 10^2, 10^3$ , and  $10^4$  tuples.

### 4.4.2 Experiments for Search Queries

We measured the running time per query while varying the number of numeric-type attributes. We take the ring modulus  $m = 8191$ , and each of the ciphertexts has 630 plaintext slots. For  $N = 1$ , the experiment of  $(\bar{Q}^*.1)$  query is given in the top three rows of Table 4.4 and that of  $(\bar{Q}^*.2)$  is in the bottom three rows in Table 4.4, where  $s$  is the number of attributes,  $L$  is the number of ciphertext moduli, and Comm. means the communication cost.

Table 4.4: Performance of  $(\bar{Q}^*.1)$  and  $(\bar{Q}^*.2)$

Message Space	$\tau$	$L$	$s$	Timing	Comm.
$\mathbb{Z}_2$	1	6	5	0.38 s	53.99 KB
			10	0.76 s	107.97 KB
			20	1.51 s	215.95 KB
$\mathbb{Z}_2$	4	7	5	2.04 s	73.48 KB
			10	4.09 s	146.96 KB
			20	8.17 s	293.93 KB

### 4.4.3 Experiments for Search-and-Sum

We conducted a series of additional experiments to measure performance of search-and-compute queries. Because each of the ciphertexts can hold  $\ell$  plaintext slots of elements in  $\mathbb{Z}_{2^r}$  and because a numeric-type attribute with

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

a length of 30 bits is encoded into  $\tilde{\omega} (= \lceil 30/\log(2^\omega) \rceil = \lceil 30/\omega \rceil)$  slots, we can process  $\tilde{\ell} (= \lfloor \ell/\tilde{\omega} \rfloor)$  attributes in each evaluation. This yields an amortized rate of running times per attributes.

At first glance, a larger  $\omega$  seems to be better. However, if  $\omega$  is too large, by Theorem 4.4.1, a plaintext modulus  $2^r$  becomes large. This results in an increased depth of circuits. Therefore, we need to choose a sufficiently large  $\omega$  whereby the resulting plaintext space is not too large.

We divided our experiment into four cases: (1) Single equality, (2) Multiple equality, (3) Single comparison, and (4) Multiple comparison.

**Case I: Single Equality.** This case contains one equality test in the **where** clause. We chose a plaintext space so that the number of plaintext slots is divisible by 10. Then, the entire keyword attribute is packed in only one ciphertext. Further, we take the ring modulus  $m$  whereby there exists  $g \in \mathbb{Z}_m^*$  that has order 10 in the original group  $\mathbb{Z}_m^*$  and in the quotient group  $\mathbb{Z}_m^*/\langle 2 \rangle$ . Then there is a Frobenius automorphism of cyclic right shift over those 10 plaintext bits. We used  $m = 13981$  so that each of the ciphertexts holds 600 plaintext slots. We report this experimental result in Table 4.5.

Table 4.5: Experiments for case I ( $\bar{Q}^*.3$ )

$N$	$\varepsilon$	Message Space	Radix	$L$	Timing	Comm.
$10^2$	$< 16\%$	$\mathbb{Z}_{2^{14}}$	$2^{10}$	14	3.69s	3.47KB
	$< 32\%$	$\mathbb{Z}_{2^{15}}$		15	3.89s	3.75KB
$10^3$	$\leq 6\%$	$\mathbb{Z}_{2^{16}}$	$2^{10}$	15	38.78s	3.75KB
	$\leq 25\%$		$2^8$		51.64s	5.01KB
$10^4$	$\leq 10\%$	$\mathbb{Z}_{2^{16}}$	$2^6$	15	681.05s	6.25KB
	$\leq 20\%$		$2^5$		817.26s	7.50KB
	$\leq 40\%$		$2^4$		1089.68s	10.03KB

**Case II: Multiple Equality.** This case contains two or more equality tests in the **where** clause (*i.e.*,  $\tau \geq 2$ ). We performed experiments for  $\tau =$

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

2 and  $\tau = 4$ . When  $\tau = 2$ , we used  $m = 13981$  as before. For the  $\tau = 4$  case, we chose  $m = 20485$  to support more multiplications than before. Similarly, each ciphertext holds 640 plaintext slots. Compared with queries in the conjunctive form, disjunctive-formed queries require more addition operations. However, both of them require the same depth; therefore, their running times are not significantly different from each other.

Each result is presented in Table 4.6 and Table 4.7 (the 6th column of each table consists of two parts: The left part is for conjunctive-formed queries, and the right part is for disjunctive-formed ones.)

Table 4.6: Experiments for case II ( $\tau = 2$ )

$N$	$\varepsilon$	Message Space	Radix	$L$	Timing		Comm.
$10^2$	$< 16\%$	$\mathbb{Z}_{2^{14}}$	$2^{10}$	16	4.81s	4.84s	3.68KB
	$< 32\%$	$\mathbb{Z}_{2^{15}}$		17	5.12s	5.26s	3.98KB
$10^3$	$\leq 6\%$	$\mathbb{Z}_{2^{16}}$	$2^{10}$	17	51.63s	52.14s	3.98KB
	$\leq 25\%$		$2^8$		68.83s	69.52s	5.31KB
$10^4$	$\leq 10\%$	$\mathbb{Z}_{2^{16}}$	$2^6$	17	913.18s	926.11s	6.64KB
	$\leq 20\%$		$2^5$		1095.81s	1111.33s	7.97KB
	$\leq 40\%$		$2^4$		1261.08s	1481.77s	10.63KB

Table 4.7: Experiments for case II ( $\tau = 4$ )

$N$	$\varepsilon$	Message Space	Radix	$L$	Timing		Comm.
$10^2$	$< 16\%$	$\mathbb{Z}_{2^{14}}$	$2^{10}$	18	9.79s	9.86s	5.09KB
	$< 32\%$	$\mathbb{Z}_{2^{15}}$		19	10.24s	10.28s	5.44KB
$10^3$	$\leq 6\%$	$\mathbb{Z}_{2^{16}}$	$2^{10}$	19	101.86s	105.15s	5.44KB
	$\leq 25\%$		$2^8$		135.59s	139.97s	7.24KB
$10^4$	$\leq 10\%$	$\mathbb{Z}_{2^{16}}$	$2^6$	19	1788.19s	1800.84s	9.05KB
	$\leq 20\%$	$\mathbb{Z}_{2^{17}}$	$2^6$	20	1850.70s	1864.36s	9.05KB
	$\leq 40\%$	$\mathbb{Z}_{2^{17}}$	$2^5$	20	2234.81s	2251.30s	10.93KB

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

**Case III: Single Comparison.** This case contains one greater-than comparison in the **where** clause. For the experiments, we used  $m = 20485$  in the case of  $L = 20$ , but in all other experiments, we used  $m = 13981$ . We report the experimental results in Table 4.8.

Table 4.8: Experiments for case III

$N$	$\varepsilon$	Message Space	Radix	$L$	Timing	Comm.
$10^2$	$< 16\%$	$\mathbb{Z}_{2^{14}}$	$2^{10}$	17	9.98s	3.71KB
	$< 32\%$		$2^9$		13.31s	4.94KB
$10^3$	$\leq 6\%$	$\mathbb{Z}_{2^{14}}$	$2^8$	17	133.12s	4.94KB
	$\leq 25\%$		$2^6$		166.40s	6.18KB
$10^4$	$\leq 10\%$	$\mathbb{Z}_{2^{14}}$	$2^4$	17	2805.97s	9.88KB
	$\leq 20\%$	$\mathbb{Z}_{2^{17}}$	$2^6$	20	3116.66s	10.66KB
	$\leq 40\%$	$\mathbb{Z}_{2^{17}}$	$2^5$	20	3763.51s	12.88KB

We observed that the results for Case IV are very similar to those for Case II. Thus, due to space limitations, we omitted the Case IV experimental results. For a better comparison, in Figure 4.2, we graphically depict the experimental results described above, while the selectivity ratio  $\varepsilon$  is fixed at 10%.

### 4.4.4 Experiments for Search-and-Count

The experiments for search-and-count can also be divided into four cases as performed above. In these experiment, the plaintext modulus  $m = 13981$  was used; therefore, each of the ciphertexts holds 600 plaintext slots. Table 4.9 shows the case with a single equality condition, Table 4.10 shows that with  $\tau = 4$ , and Table 4.11 shows that with a single comparison condition.

Finally, we summarize the above experiments using the graph presented in Figure 4.3, where we have also fixed the selectivity ratio at 10%.



## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

Figure 4.2: Experimental results for search-and-sum

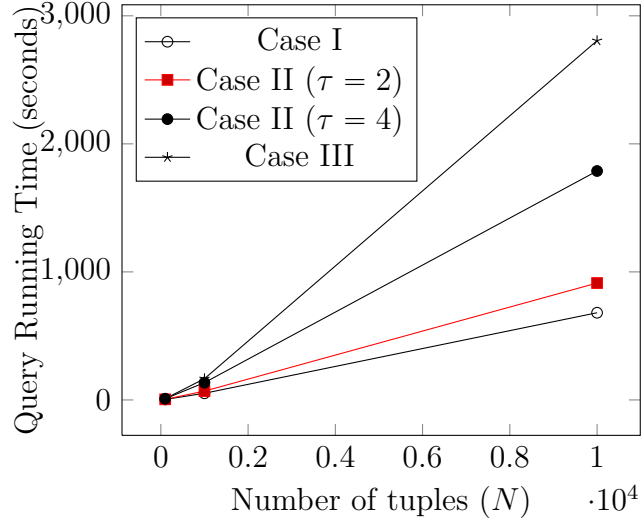


Table 4.9: Experiments using single equality

$N$	$\varepsilon$	Message Space	$L$	Timing	Comm.
$10^2$	$< 8\%$	$\mathbb{Z}_{2^3}$	7	5.66s	0.73KB
	$< 32\%$	$\mathbb{Z}_{2^5}$	8	7.34s	1.00KB
$10^3$	$\leq 6\%$	$\mathbb{Z}_{2^6}$	10	84.59s	0.93KB
	$\leq 25\%$	$\mathbb{Z}_{2^8}$	11	90.89s	1.03KB
$10^4$	$\leq 40\%$	$\mathbb{Z}_{2^{12}}$	12	961.84s	1.12KB

Table 4.10: Experiments for multiple equality ( $\tau = 4$ )

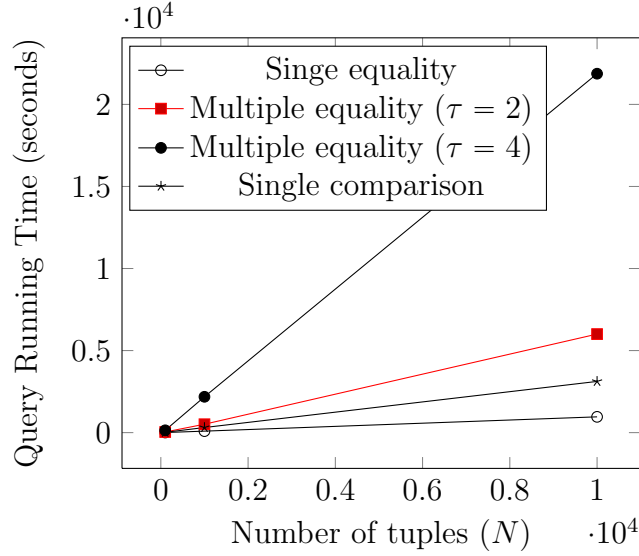
$N$	$\varepsilon$	Message Space	$L$	Timing		Comm.
$10^2$	$< 8\%$	$\mathbb{Z}_{2^3}$	9	131.35s	132.14s	0.91KB
	$< 32\%$	$\mathbb{Z}_{2^5}$	10	142.28s	144.13s	1.03KB
$10^3$	$\leq 6\%$	$\mathbb{Z}_{2^6}$	12	1718.08s	1741.13s	1.22KB
	$\leq 25\%$	$\mathbb{Z}_{2^8}$	15	2184.16s	2178.22s	1.23KB
$10^4$	$\leq 40\%$	$\mathbb{Z}_{2^{12}}$	16	21870.80s	22195.40s	1.25KB

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

Table 4.11: Experiments using Single Comparison

$N$	$\varepsilon$	Message Space	$L$	Timing	Comm.
$10^2$	$< 8\%$	$\mathbb{Z}_{2^3}$	8	17.10s	0.82KB
	$< 32\%$	$\mathbb{Z}_{2^5}$	9	19.24s	0.91KB
$10^3$	$\leq 6\%$	$\mathbb{Z}_{2^6}$	11	224.04s	0.93KB
	$\leq 25\%$	$\mathbb{Z}_{2^8}$	15	311.84s	1.25KB
$10^4$	$\leq 40\%$	$\mathbb{Z}_{2^{12}}$	15	3029.05s	1.25KB

Figure 4.3: Experimental results for search-and-count



### 4.5 Handling Join Query

In this section, we design the join queries within the search-and-compute paradigm. For this purpose, suppose that we have the other relation  $S(B_1, \dots, B_e)$  consisting of  $M$  tuples. For simplicity, we assume that  $N \geq M$ .

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

First, we consider a simple join query as follows:

$$\begin{aligned}
 &\text{select } r.A_{j_1}, \dots, r.A_{j_s}, s.B_{j'_1}, \dots, s.B_{j'_{s'}} \\
 &\text{from } R \text{ as } r, S \text{ as } s \\
 &\text{where } r.A_{j_k} = s.B_{j'_{k'}};
 \end{aligned} \tag{Q.4}$$

This type of query is expressed and efficiently processed using only the equality circuit. Specifically, for each  $i \in [1, N], i' \in [1, M]$ , the query  $(\bar{Q}^*.4)$  is expressed as

$$\text{equal} \left( r.\bar{A}_{j_k}^{(i)}, s.\bar{B}_{j'_{k'}}^{(i')} \right) \cdot \left( r.\bar{A}_{j_1}^{(i)}, s.\bar{B}_{j'_1}^{(i')}, \dots \right) \tag{\bar{Q}^*.4}$$

For fixed  $i$  and  $i'$ , we suppose that each numeric-type attribute is packed in only one ciphertext. Then, the only difference from  $(\bar{Q}^*.1)$  is that  $(\bar{Q}^*.4)$  requires two homomorphic multiplications by the result of search operations; thus, we need to perform  $NM$  equality tests in total. Hence, the depth of circuit needed to process  $(\bar{Q}^*.4)$  is  $1 + \log \mu$ , and the computation complexity is  $(2NM) \text{ HA} + NM(2 + \log \mu) \text{ HM}$ . In addition, a join query with  $\tau$  conjunctive conditions needs to perform  $\tau NM$  equality tests; therefore, the required depth is  $(1 + \log \mu + \log \tau)$ .

Next, we consider an advanced join query demanding search-and-compute operations. For the sake of readability, we consider a join query with two aggregate functions and a simple condition as follows.

$$\begin{aligned}
 &\text{select } \text{sum}(r.A_j), \text{count}(*) \\
 &\text{from } R \text{ as } r, S \text{ as } s \\
 &\text{where } r.A_{j_k} = s.B_{j'_{k'}};
 \end{aligned} \tag{Q.5}$$

We can express the query  $(\bar{Q}^*.5)$  in a manner similar to that used in Section 4.2.2. Assuming  $\text{sum}(r.A_j) < 2^\mu NM$ , we use a full adder of size  $\nu = \mu + \log(NM)$ . By contrast, the result of  $\text{count}(<) < NM$ , and it suffices to use a full adder of size  $\log(NM)$ . Thus, one candidate of circuit construction

## CHAPTER 4. PRIVATE DATABASE QUERY PROCESSING

for  $(\bar{Q}^*.5)$  is as follows:

$$\begin{aligned} & \text{fadd}_{\mu + \log NM} \left( \text{equal} \left( r.\bar{A}_{j_k}^{(i)}, s.\bar{B}_{j_{k'}}^{(i')} \right) \cdot r.\bar{A}_j^{(i)} \right), \\ & \text{fadd}_{\log NM} \left( \text{equal} \left( r.\bar{A}_{j_k}^{(i)}, s.\bar{B}_{j_{k'}}^{(i')} \right) \right). \end{aligned} \quad (\bar{Q}^*.5)$$

With respect to  $\text{sum}(r.A_j)$ , this is the same as  $(\bar{Q}^*.3)$ , except for the number of operands for additions. Therefore, the depth for evaluation amounts to  $1 + \log \mu + \log(NM)(1 + \log(\nu - 2))$ , and the computation complexity is  $(2NM + \nu(NM - 1))\text{HA} + (NM(1 + \log \mu) + (NM - 1)(3\nu - 5))\text{HM}$ .

We remark that it is straightforward to extend this approach to a join query with two or more aggregate functions in the **select** clause.

Finally, we performed some experiments for join queries. We measured the average running time for processing a single equality test while varying  $N, M$  from 10 to  $10^2$  and fixing  $s = s' = 5$ . Table 4.12 reports the experimental results for  $(\bar{Q}^*.4)$ . As  $N$  and  $M$  increase, the running time of the algorithm grows linearly. Table 4.13 shows the experimental results for  $(\bar{Q}^*.5)$ , assuming the selectivity ratio is fixed at 10%. Because the experiments of  $(\bar{Q}^*.4)$  are implemented for more numeric-type attributes than those for  $(\bar{Q}^*.5)$ , the query takes longer to perform.

Table 4.12: Experiments for  $(\bar{Q}^*.4)$

$\tau = 1$ and $s = s' = 5$		
$N = M = 10$	$N = 10^2, M = 10$	$N = M = 10^2$
42.75 s	423.86 s	4210.53 s

Table 4.13: Experiments for  $(\bar{Q}^*.5)$

$\tau = s = 1$		
$N = M = 10$	$N = 10^2, M = 10$	$N = M = 10^2$
3.79 s	50.84 s	680.27 s

## Chapter 5

# Secure Genome Analysis Based on Homomorphic Encryption

### 5.1 Exact Edit Distance Algorithm

We now describe how to execute the homomorphic computation of the exact edit distance algorithm using the primitive circuits. Then we analyze the performance of our encrypted edit distance algorithm.

Let  $|\Sigma|$  be the size of a alphabet and denote  $\omega = \lceil \log |\Sigma| \rceil$ . As mentioned before, let  $\alpha$  and  $\beta$  be two strings over  $\omega$ -bit alphabet. Then each character of the strings can be seen as an  $\omega$ -bit value.

#### 5.1.1 Encrypted Edit Distance Algorithm

Since all the values  $D(i, j)$ 's are less than  $n + m - 1$ , we may assume that they are  $\lceil \log(n + m - 1) \rceil$ -bits, say  $\mu$ . Suppose that we have computed  $D(i - 1, j - 1)$ ,  $D(i, j - 1)$ ,  $D(i - 1, j)$ , and  $\omega$ -bit characters  $\alpha_i$  and  $\beta_j$ . From the

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

fact that  $\text{sub}(\alpha_i, \beta_j) = \text{equal}(\alpha_i, \beta_j) \oplus 1$ , we know

$$\begin{aligned}
 (D_{i-1,j-1} + \text{sub}(\alpha_i, \beta_j))[k] &= ((\text{sub}(\alpha_i, \beta_j) \oplus 1) \wedge D_{i-1,j-1}[k]) \oplus \\
 &\quad (\text{sub}(\alpha_i, \beta_j) \wedge \text{ADD}(D_{i-1,j-1}, 1)[k]) \\
 &= (\text{equal}(\alpha_i, \beta_j) \wedge D_{i-1,j-1}[k]) \oplus \\
 &\quad ((\text{equal}(\alpha_i, \beta_j) \oplus 1) \wedge \text{ADD}(D_{i-1,j-1}, 1)[k])
 \end{aligned}$$

for  $1 \leq k \leq \mu$  and

$$\text{ADD}(D_{i-1,j-1}, 1)[k] = \begin{cases} D_{i-1,j-1}[1] \oplus 1 & \text{if } k = 1, \\ D_{i-1,j-1}[k] \oplus (\wedge_{l=1}^{k-1} D_{i-1,j-1}[l]) & \text{if } 2 \leq k \leq \mu. \end{cases}$$

In the same way as in Section 3.1.4,  $\text{ADD}(D_{i-1,j-1}, 1)$  can be implemented with a SWHE scheme of depth  $\log(\mu - 1)$  in  $\mu$  homomorphic additions and  $\left(\frac{(\mu-1)\log(\mu-1)}{2} - 2\right)$  homomorphic multiplications since we only need to compute  $\prod_{l=1}^{k-1} \text{Enc}(D(i-1, j-1)[l])$ . From these observations,  $D_{i,j} = \min\{D(i-1, j-1) + t_{i,j}, D(i, j-1) + 1, D(i-1, j) + 1\}$  can be written as arithmetic circuits using the above circuits. Hence, given ciphertexts  $\text{Enc}(D(i-1, j-1))$ ,  $\text{Enc}(D(i, j-1))$ ,  $\text{Enc}(D(i-1, j))$ ,  $\text{Enc}(\alpha_i)$ , and  $\text{Enc}(\beta_j)$ , one can apply these operations so as to compute the encryption of  $D(i, j)$ . Continuing this way, we obtain the encrypted edit distance  $\text{Enc}(D(n, m))$ .

By the building block algorithms of  $\text{com}$  (in Lemma 3.1.2) and  $\text{ADD}$ , the one diagonal-round circuits have

$$\begin{aligned}
 D &= 2\log(\mu - 1) + 4, \\
 \text{HA} &= 15\mu + \omega - 6, \\
 \text{HM} &= 3(\mu - 1)\log(\mu - 1) + 11\mu + \omega - 13.
 \end{aligned}$$

It is possible to compute  $D(i, j)$ 's simultaneously when  $i + j$  is a fixed value from  $1, 2, \dots, (n + m - 1)$ , so we expect to consume  $(2\log(\mu - 1) + 4) \cdot (n + m - 1)$  levels for computing them diagonally, which requires  $(15\mu + \omega - 6)nm$  homomorphic additions and  $(3(\mu - 1)\log(\mu - 1) + 11\mu + \omega - 13)nm$

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

multiplications in total. In other words, given two encrypted sequences of lengths  $n$  and  $m$ , a SWHE scheme of depth  $O((n + m) \log(\log(n + m)))$  can evaluate the edit distance algorithm in  $O(nm \log(n + m))$  homomorphic computations.

### 5.1.2 Optimizations Based on Block Computation

The method described in the previous subsection is computation intensive, as it needs to evaluate all  $\text{Enc}(D(i, j))$  for  $0 \leq i \leq n, 0 \leq j \leq m$  to obtain the encrypted edit distance  $\text{Enc}(D(n, m))$ . We now present an optimization to reduce the depth during the homomorphic evaluations of the algorithm.

Let us consider the  $3 \times 3$  block  $B$  in Figure 5.1. It is true that if we have computed the top and left values of this block,  $D(i-2, j-2)$ ,  $D(i-2, j-1)$ ,  $D(i-2, j)$ ,  $D(i-1, j-2)$ ,  $D(i, j-2)$ , then all other values can be expressed in terms of them. For example,  $D(i, j)$  is the minimum value between the following 7 numbers:  $D(i-2, j-2) + \text{sub}(\alpha_{i-1}, \beta_{j-1}) + \text{sub}(\alpha_i, \beta_j)$ ,  $D(i-2, j-1) + \text{sub}(\alpha_{i-1}, \beta_j) + 1$ ,  $D(i-2, j) + \text{sub}(\alpha_i, \beta_j) + 1$ ,  $D(i-1, j-2) + \text{sub}(\alpha_i, \beta_{j-1}) + 1$ ,  $D(i-1, j-1) + \text{sub}(\alpha_i, \beta_j) + 1$ ,  $D(i-1, j) + 2$ ,  $D(i, j-2) + 2$ .

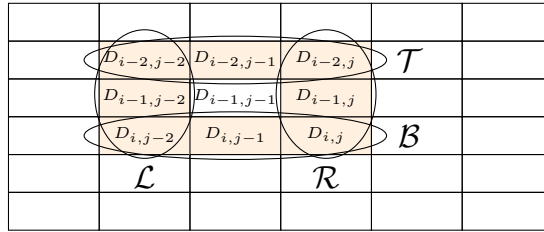


Figure 5.1: Block of size 3

In general, we consider a block of size- $(\tau + 1)$  which consists of the following sets:

$$\begin{aligned}
 \text{top} & : \mathcal{T} = \{D(i - \tau, j - \tau), D(i - \tau, j - \tau + 1), \dots, D(i - \tau, j)\}, \\
 \text{left} & : \mathcal{L} = \{D(i - \tau, j - \tau), D(i - \tau + 1, j - \tau), \dots, D(i, j - \tau)\}, \\
 \text{right} & : \mathcal{R} = \{D(i - \tau, j), D(i - \tau + 1, j), \dots, D(i, j)\}, \\
 \text{bottom} & : \mathcal{B} = \{D(i, j - \tau), D(i, j - \tau + 1), \dots, D(i, j)\}.
 \end{aligned}$$

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

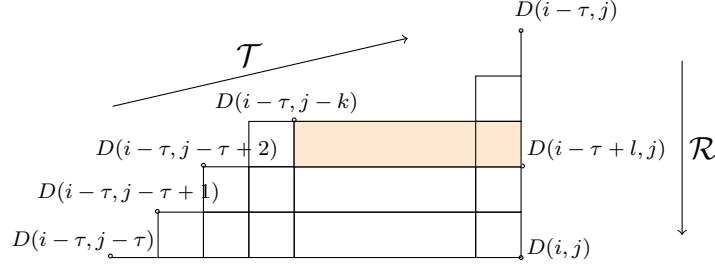


Figure 5.2: Grid of size-( $\tau + 1$ ) block

Then all the values of  $\mathcal{R}$  and  $\mathcal{B}$  are expressed in terms of values of  $\mathcal{T}$  and  $\mathcal{L}$ .

More precisely, consider the grid shown in Figure 5.2. It can be obtained by a rotation by 45 degree clockwise from the initial Edit distance block. One can only move one unit right or down on the grid: if moving right from  $D(i - k, j - l)$ , then  $\text{sub}(i - k + 1, j - l + 1)$  is added to the value and we obtain  $D(i - k, j - l) + \text{sub}(i - k + 1, j - l + 1)$ . In the case of moving one unit down, “1” is added to it. We note that the number of shortest paths from  $D(i - \tau, j - k)$  to  $D(i - \tau + l, j)$  is  $\frac{l!}{k!(l-k)!} = \binom{l}{k}$  for some  $l \geq k$  since the paths include  $k$  steps in the  $x$  axis and  $(l - k)$  steps in  $y$  axis. It is seen as the the number of the functions of  $D(i - \tau + l, j)$  in terms of  $D(i - \tau, j - k)$ . From these observations,  $D(i - \tau + l, j)$  is the minimum between  $\sum_{k=0}^l \binom{l}{k} = 2^l$  values. In particular,  $D(i, j)$  is the minimum between  $2 \cdot \sum_{k=0}^{\tau} \binom{\tau}{k} - \tau = 2^{\tau+1} - \tau$  values because the set of all the paths of  $D(i, j)$  is symmetric with respect to the line from  $D(i - \tau, j - \tau)$  to  $D(i, j)$ . We know that the minimum circuits consume the largest number of levels than others (equality circuit or addition circuits), and it needs  $O(\log k)$  levels to evaluate the minimum circuits  $\min^k$  that compute the minimum value between  $k$  numbers, which requires  $O(k^2)$  homomorphic computations. Thus, one can compute a block of size-( $\tau + 1$ ) by evaluating the circuits with a SWHE of depth  $O(\log(2^{\tau+1} - \tau)) \approx O(\tau)$  in

$$\sum_{k=2, 2^2, \dots, 2^{\tau-1}} O(k^2) + O((2^{\tau+1} - \tau)^2) \approx O(2^{2\tau})$$

homomorphic operations. From the fact that all the blocks of size-( $\tau + 1$ )



## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

can be computed diagonally while shares of the values of  $\mathcal{T}$  and  $\mathcal{L}$  have been computed, we can conclude that the edit distance algorithm can be implemented using  $O(2^{2\tau} \cdot \frac{nm}{\tau^2})$  homomorphic operations with a SWHE scheme of depth  $O(\tau \cdot (\frac{n+m}{\tau} - 1)) \approx O(n + m)$  for given two encrypted sequences of lengths  $n$  and  $m$ . Hence, this optimization reduces the depth but the entire computation increases as  $\tau$  becomes larger.

### 5.1.3 Optimization of Encrypted Edit Distance Algorithm Based on Pathfinding Method

When the sizes of two DNA sequences are the same, we can propose an optimized method based on pathfinding to effectively reduce the circuit depth compared with the proposed methods.

The calculation of edit distance is equivalent to find a path from  $(0, 0)$  to  $(n, n)$  that achieves minimum accumulated cost for insertions, deletions and substitutions. Denote  $\mathcal{P}^l(n) = \{\text{op}_1^l(\alpha_1^l, \beta_1^l), \text{op}_2^l(\alpha_2^l, \beta_2^l), \dots, \text{op}_{k_l}^l(\alpha_{k_l}^l, \beta_{k_l}^l)\}$  as a set of operations in the  $l$ -th candidate path, where an operation can be  $\text{op}_i^l(\alpha_i^l, \beta_i^l) \in \{\text{sub}(\alpha_i^l, \beta_i^l), \text{ins}(\beta_i^l), \text{del}(\alpha_i^l)\}$ . For simplicity, we omit the index  $l$  in the following. The transitions of indices and costs due to the substitution operation  $\text{sub}(\alpha_i, \beta_i)$  can be defined by

$$\begin{aligned} \alpha_i &= \alpha_{i-1} + 1, \quad \beta_i = \beta_{i-1} + 1, \\ D(\alpha_i, \beta_i) &= D(\alpha_{i-1}, \beta_{i-1}) + \text{sub}(\alpha_i, \beta_i). \end{aligned}$$

Similarly, we define the insertion operation  $\text{ins}(\beta_i)$  as

$$\alpha_i = \alpha_{i-1}, \quad \beta_i = \beta_{i-1} + 1, \quad D(\alpha_i, \beta_i) = D(\alpha_{i-1}, \beta_{i-1}) + 1,$$

and deletion operation  $\text{del}(\alpha_i)$  as

$$\alpha_i = \alpha_{i-1} + 1, \quad \beta_i = \beta_{i-1}, \quad D(\alpha_i, \beta_i) = D(\alpha_{i-1}, \beta_{i-1}) + 1.$$

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

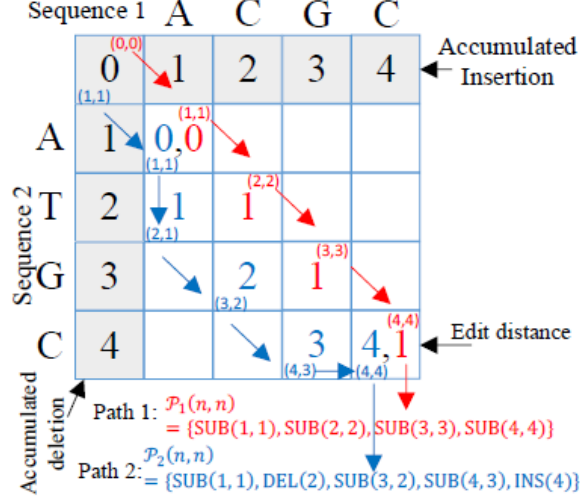


Figure 5.3: An example of pathfinding based edit distance computation

Figure 5.3 shows an example of pathfinding based edit distance computation where we included two different paths. The blue path  $\mathcal{P}^2(n)$  is represented as

$$\{\text{sub}(\alpha_1, \beta_1), \text{del}(\alpha_2), \text{sub}(\alpha_3, \beta_2), \text{sub}(\alpha_4, \beta_3), \text{ins}(\beta_4)\},$$

so the corresponding candidate for  $D(n, n)$  is

$$\text{sub}(\alpha_1, \beta_1) + 1 + \text{sub}(\alpha_3, \beta_2) + \text{sub}(\alpha_4, \beta_3) + 1.$$

In this example, the path in red color yields the minimum costs.

In general, a path  $\mathcal{P}(n)$  can be further represented by a set of segments, where each segment ends at the position with  $\alpha_i = \beta_i$ :

$$\mathcal{P}(n) = \{\mathcal{P}^0(s_0, \sigma_0, \theta_0), \mathcal{P}^1(s_1, \sigma_1, \theta_1), \dots, \mathcal{P}^r(s_r, \sigma_r, \theta_r)\}$$

where  $\mathcal{P}^i(s_i, \sigma_i, \theta_i)$  is the  $i$ -th segment between  $(\alpha_{s_i}, \beta_{s_i})$  and  $(\alpha_{s_{i+1}-1}, \beta_{s_{i+1}-1})$  that contains  $\theta_i$  insertion and deletion operations, respectively. Here,  $r$  is the number of segments and  $s_i$  is the first starting index of the coordinates

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

$(\alpha_{s^i}, \beta_{s^i})$  in the  $i$ -th segment. The parameter  $\sigma_i$  indicates the location of the segment in the matrix as 0 for diagonal, 1 for lower triangular, and  $-1$  for upper triangular. For example, the blue path  $\mathcal{P}^2(n)$  is divided into two segments:  $\mathcal{P}^0(s_0 = 0, \sigma_0 = 0, \theta_0 = 0) = \{\text{sub}(\alpha_1, \beta_1)\}$  and  $\mathcal{P}^1(s_1 = 1, \sigma_1 = 1, \theta_1 = 1) = \{\text{del}(\alpha_2), \text{sub}(\alpha_3, \beta_2), \text{sub}(\alpha_4, \beta_3), \text{ins}(\beta_4)\}$ . Thus we can derive the candidate cost  $D_l(n, n)$  for the path  $\mathcal{P}_l(n)$  as follows:

$$\begin{aligned} D^l(n, n) &= \sum_{i=1}^r D^l(\mathcal{P}_i^l(s_i, \sigma_i, \theta_i)) = \sum_{i=1}^r (2\theta_i^l + \text{sub}(\alpha_{s_i}, \beta_{s_i})) \\ &= 2\theta^l + \text{sub}(\alpha_{s^1}^l, \beta_{s^1}^l) + \cdots + \text{sub}(\alpha_{n-\theta^l}^l, \beta_{n-\theta^l}^l) \end{aligned}$$

where  $D^l(\mathcal{P}_i^l(s_i, \sigma_i, \theta_i))$  is the derived cost for segment  $\mathcal{P}_i^l(s_i, \sigma_i, \theta_i)$  and  $\theta^l = \sum_{i=1}^r \theta_i^l$ . Therefore, the edit distance is the minimum of costs for all candidate paths, that is,  $D(n, n) = \min_{0 \leq l \leq \Omega_n} (D_l(n, n))$  where  $\Omega_n$  denotes the number of candidate paths.

**Remark 5.1.1.** Since  $D(n, n)$  is less than  $n$  and  $D(i, 0), D(0, i)$  is greater than  $2i$ , we can find each candidate path  $\mathcal{P}_l$  is constrained by  $|\alpha_i^l - \beta_i^l| \leq \lfloor \frac{n}{2} \rfloor - 1$  and  $\theta_l \leq \lfloor \frac{n}{2} \rfloor - 1$ . Moreover, for arbitrary  $\mathcal{P}_i^l(s_i, \sigma_i, \theta_i)$ , the edit distance of a block with the diagonal from  $(\alpha_{s_i}^l, \beta_{s_i}^l)$  to  $(\alpha_{s_{i+1}-1}^l, \beta_{s_{i+1}-1}^l)$  is not greater than  $\alpha_{s_{i+1}-1}^l - \beta_{s_i}^l$ , that is,  $2\theta_i^l < \alpha_{s_{i+1}-1}^l - \beta_{s_i}^l$ . This means that all the segments  $\mathcal{P}_i^l(s_i, \sigma_i, \theta_i)$  with  $\alpha_{s_{i+1}-1}^l - \beta_{s_i}^l \leq 2\theta_i^l$  shall be excluded from  $\mathcal{P}^l$ .

Algorithm 3 describes our protocol based on path-based optimization. Given the  $\Omega_n$  candidate paths, Algorithm 3 computes the cost  $\text{Enc}(D_i(n, n))$  for the path  $\mathcal{P}_l$  from  $D(0, 0) = 0$ . For possible edit distance  $d \in \{0, 1, \dots, n\}$ , it evaluates the arithmetic circuit  $\text{equal}(\prod_l (D^l(n) - d), 0)$ , which has the value 1 if and only if  $D^l(n) = d$ . For simplicity, we denote

$$\mathcal{F}^d = \text{equal}\left(\prod_l (\text{Enc}(D^l(n)) - d), 0\right)$$

as the encrypted flag of verification. Finally, the protocol outputs the set of flags  $\mathcal{F} = \{\mathcal{F}^d, d = 0, 1, \dots, n\}$ . Let us denote  $e_d$  the decrypted value of the

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

---

### Algorithm 3 Edit Distance Algorithm Based on Pathfinding Method

---

```

1: for each path  $l = 1, 2, \dots, \Omega_n$  do
2:   Compute the cost  $\text{Enc}(D^l(n))$  for path  $\mathcal{P}^l$ 
3:    $\text{Enc}(D^l(n)) \leftarrow 2\text{Enc}(\theta^l) + \text{Enc}(\text{sub}(\alpha_{i_1}^l, \beta_{i_1}^l)) + \dots + \text{Enc}(\text{sub}(\alpha_{i_{n-\theta^l}}^l, \beta_{i_{n-\theta^l}}^l))$ 
4: end for
5: for possible edit distance  $d = 0, 1, \dots, n$  do
6:    $\mathcal{F}^d = \text{equal}(\prod_l (\text{Enc}(D^l(n)) - d), 0)$ 
7: end for
8: return  $\mathcal{F} = \{\mathcal{F}^d, d = 0, 1, \dots, n\}$ 

```

---

resulting ciphertext. Then the edit distance is the smallest integer  $d$  such that  $e_d = 1$ .

#### 5.1.4 Implementation

In the following we give an estimated performance of the encrypted edit distance algorithm over DNA sequences and provide concrete timings for homomorphic evaluation of the algorithm with Shoup's NTL library [S<sup>+</sup>01] and Halevi-Shoup's HE library [HS13] over GMP. The evaluations were made on an Ubuntu 14.04 server with Intel Xeon CPU E5-2687W @ 3.10GHz and 256 GB memory. In order to check the accuracy and efficiency of our protocol, pairs of sequences  $(\alpha, \beta)$  with various lengths  $n, m$  were randomly generated for edit distance calculation.

In our scenario, the third parties first partition their own DNA sequences into segments of length  $n$  or  $m$ . Then each of the DNA sequences is encrypted as a different ciphertext with a homomorphic encryption scheme. For parallel computation, we use an encryption scheme with plaintext space  $\mathbb{Z}_t^\ell$  supporting SIMD operations with  $\ell$  slots. Then one party sends the ciphertexts which hold the  $\ell$  segments together to a cloud. Finally, the cloud service computes the edit distances of  $\ell$  different sequence pairs simultaneously. The amortized time is computed as the total time of this algorithm evaluation divided by  $\ell$ .

In our implementation, we used the optimization techniques as described

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

in Section 5.1.3, especially, the equality circuit described in Section 3.2.3:

$$\text{equal}(x, y) = \mathbf{1}_0(x - y) \equiv M_0 \cdot \prod_{i=1}^{t-1} (x - i) \pmod{t}$$

for some constant  $M_0$ . From the Wilson's Theorem, it always holds for a prime number  $t$  greater than 2 that  $(t - 1)! \equiv -1 \pmod{t}$ , so we see that  $M_0 \equiv -1 \pmod{t}$ .

The implementation results with 80-bits of security are described in Table 5.1. Here,  $\Omega_n$  is the number of candidate paths satisfying the conditions of Remark 5.1.1. The plaintext modulus  $t$  is set to the minimal prime number greater than  $n$ . Consequently, the number of levels  $L$  in modulus chain and the number of slots  $\ell$  for parallel computation also vary with  $n$ . For example, it takes 7.38 seconds to obtain the encrypted edit distance from the two encrypted DNA sequences of length 10.

Table 5.1: Implementation results of edit distance based on pathfinding

$n$	$t$	$L$	$\ell$	$\Omega_n$	KeyGen	Encrypt	Evaluation	Amortized
3	5	7	390	3	8.299s	1.072s	3.812s	0.009s
4		8	240	7	8.776s	1.294s	8.577s	0.036s
5	7	10	180	19	11.786s	1.807s	25.038s	0.139s
6		11	130	47	14.499s	2.299s	54.187s	0.417s
7	11	13	300	127	18.270s	3.382s	158.273s	0.528s
8		15	1436	327	20.340s	5.005s	430.512s	0.299s
9		16	792	887	31.295s	9.020s	2311.190s	2.912s
10		17	838	1719	37.812s	10.723s	6187.170s	7.383s

## 5.2 Approximate Edit Distance Algorithm

In this section, we describe how to encode and encrypt the genomic data. Based on these methods, we propose the evaluation algorithms to compute the genomic tests on encrypted data.

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

### 5.2.1 Encoding Genomic Data

The first step is to curate the data using the positions in the VCF files of two participants. In other words, the server should arrange the information and make the merged list  $\mathcal{L}$  so that each individual can encode their genotypic information according to the list. Recall that  $\text{pos}_i$  means the reference position and  $\text{sv}_i$  the type of structural variant relative to the reference. Let  $\ell(\mathcal{L})$  denote the length of the list  $\mathcal{L}$ . Then, for  $1 \leq i \leq \ell(\mathcal{L})$ , we define two values

$$e_i = \begin{cases} 1 & \text{if } \text{pos}_i \in \mathcal{L}, \\ 0 & \text{o.w.,} \end{cases} \quad f_i = \begin{cases} 0 & \text{if } \text{sv}_i \in \{\text{INS}, \text{DEL}\}, \\ 1 & \text{o.w.,} \end{cases}$$

The value  $e_i$  defines whether the genotype at the specified locus is missing; the value  $f_i$  specifies the variants compared with the reference.

Since both VCF files are aligned with the same reference genome, we don't need to compare the columns of 'REF'. To improve performance, we assume that it suffices to compare 7 SNPs between two non-reference sequences. In the following, we describe how to encode the sequences. Each SNP is represented by two bits as  $A \rightarrow 00$ ,  $T \rightarrow 01$ ,  $G \rightarrow 10$ ,  $C \rightarrow 11$ , and then concatenated with each other. Next we pad with 1 at the end of the bit string to represent the length of SNPs and distinguish 'A' from an empty SNP. Finally, we pad with zeros to make it a binary string of length 15, denoted by  $\mathbf{s}_i$ . Let  $\mathbf{s}_i[j]$  denote  $j$ -th bit of  $\mathbf{s}_i$ . If a person's SNP at the given locus is not known (*i.e.*,  $e_i = 0$ ), it is encoded as 0-string. For example, 'TCG' is encoded as a bit string  $01||11||10||10 \dots 0$  of length 15, which can be represented as an integer of 94.

Finally let us consider the  $i$ -th genotype lengths  $D_i, D'_i$  of two participants defined as follows: when it has no variants at the given locus of the sequence, set zero as the length at the locus. If it includes a deletion compared with the reference, use the length of reference. Otherwise, we take the length of the target sequence at the current locus. In Figure 5.4, we illustrate the file format of the data and its encodings.

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

##fileformat=VCF4.2 of hu604D39										##fileformat=VCF4.2 of hu661AD0										
#CHROM	POS	ID	REF	ALT	SVTYPE	L	e	f	s	#CHROM	POS	ID	REF	ALT	SVTYPE	L	e	f	s	
1	101088593	rs4908887	C	T	SNP	101088593	1	1	1110000000000000	1	101088593	rs4908887	C	T	SNP	101088593	1	1	1110000000000000	
1	101265309	rs74183250	C	T	SNP	101265309	1	1	1110000000000000	1	101265309	rs74183250	C	T	SNP	101265309	1	1	1110000000000000	
						101371397	0	1	0000000000000000	1	101371397	rs76337964	G	T	SNP	101371397	1	1	1110000000000000	
						101459624	0	1	0000000000000000	1	101459624	rs6675353	A	T	SNP	101459624	1	1	1110000000000000	
1	10165300	rs12076922	T	G	SNP	10165300	1	1	0110000000000000	1	10165300					10165300	0	1	0000000000000000	
1	101880246	rs7556208	A	G	SNP	101880246	0	1	0000000000000000	1	101880246					101880246	0	1	0000000000000000	
						10243500	0	1	0000000000000000	1	10243500			T	INS	10243500	1	0	1110000000000000	
						10269774	0	1	0000000000000000	1	10269774	rs76680928	T	C	SNP	10269774	1	1	0110000000000000	
						102677369	1	1	1110000000000000	1	102677369	rs1938356	A	G	SNP	102677369	1	1	0110000000000000	
						102824572	0	1	0000000000000000	1	102824572	rs7553905	A	C	SNP	102824572	1	1	0110000000000000	
						102923499	0	1	0000000000000000	1	102923499					102923499	1	1	1010000000000000	
1	103529636	rs76751471	C	T	SNP	103529636	1	1	1110000000000000	1	103529636					103529636	0	1	0000000000000000	
1	103590697	rs12126095	G	T	SNP	103590697	1	1	1110000000000000	1	103590697	rs12126095	G	T	SNP	103590697	1	1	1110000000000000	
1	103763251	rs4007992	A	G	SNP	103763251	1	1	0110000000000000	1	103763251					103763251	0	1	0000000000000000	
						103774240	0	1	0000000000000000	1	103774240			G	A	SNP	103774240	1	0	0110000000000000
						104455760	0	1	0000000000000000	1	104455760					104455760	0	1	0000000000000000	
						104529308	0	1	0000000000000000	1	104529308	rs115597242	C	T	SNP	104529308	1	1	1110000000000000	
1	105179694	rs6679685	T	C	SNP	105179694	1	1	0110000000000000	1	105179694					105179694	0	1	0000000000000000	
						105274378	0	1	0000000000000000	1	105274378	rs7545116	G	T	SNP	105274378	1	1	1110000000000000	
						105397179	0	1	0000000000000000	1	105397179	rs6792224	T	C	SNP	105397179	1	1	1010000000000000	
1	105491343	rs2923291	T	A	SNP	105491343	1	1	0010000000000000	1	105491343	rs2923291	T	A	SNP	105491343	1	1	0010000000000000	
1	105797318	rs74480818	A	G	SNP	105797318	1	1	0110000000000000	1	105797318					105797318	0	1	0000000000000000	
1	106270875				G	INS	106270875	1	0	0110000000000000	1	106270875				106270875	0	1	0000000000000000	
1	106286216	rs10081343	G	A	SNP	106286216	1	1	0010000000000000	1	106286216					106286216	0	1	0000000000000000	
1	106286257				C	T	SNP	1	1	1110000000000000	1	106286257				106286257	0	1	0000000000000000	
						106543173	0	1	0000000000000000	1	106543173			G	A	SNP	106543173	1	1	0010000000000000
(a)										(b)										

Figure 5.4: A snapshot of the dataset and its encodings: (a)hu604D39 and (b)hu661AD0

### 5.2.2 Secure DNA Sequence Comparison with Bit-sliced Implementation

We represent sequence comparison algorithms as binary circuits and then evaluate them over encrypted data. We use the native plaintext space of binary polynomials (*i.e.*,  $\mathcal{R}_2 = \mathbb{Z}_2[x]/(\Phi_m(x))$ ), and denote XOR and AND as  $\oplus$  and  $\wedge$ , respectively. For simplicity, you may consider the plaintext space  $\mathbb{Z}_2^\ell$  supporting batching operation with  $\ell$  slots.

For the homomorphic evaluation of Hamming distance, the genomic data of two participants, denoted by  $(e_i, f_i, s_i)$  and  $(e'_i, f'_i, s'_i)$ , are encrypted bit-wise. For example, the encryptions of  $e_i$ 's are in the form of

$$\begin{aligned} & \text{BGV.Enc}(\text{CRT}(e_1, \dots, e_\ell), \text{pk}), \\ & \text{BGV.Enc}(\text{CRT}(e_{\ell+1}, \dots, e_{2\ell}), \text{pk}), \dots, \\ & \text{BGV.Enc}(\text{CRT}(e_{\ell(\mathcal{L})/\ell+1}, \dots, e_{\ell(\mathcal{L})}, 0, \dots, 0), \text{pk}). \end{aligned}$$

This allows to compute the same function on  $\ell$  inputs at the price of one computation.

Now, we consider the comparison binary circuit (described in [CKL15]) for the secure computation of the approximate edit distance. We express an unsigned  $\mu$ -bit integer  $x$  in its binary representation and denote the  $j$ -th coordinate of  $x$  by  $x[j]$  (*i.e.*,  $x = \sum_{j=1}^{\mu} x[j] \cdot 2^{j-1}$ ,  $x[j] \in \{0, 1\}$ ). For two  $\mu$ -bit

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

integers  $x$  and  $y$ , the comparison circuit is defined by

$$\text{com}(x, y) = \begin{cases} 1 & \text{if } x < y, \\ 0 & \text{o.w.,} \end{cases}$$

and this is written recursively as  $\text{com}(x, y) := c_\mu$  where

$$c_j = ((x[j] \oplus 1) \wedge y[j]) \oplus ((x[j] \oplus 1 \oplus y[j]) \wedge c_{j-1})$$

for  $j \geq 2$  with an initial value  $c_1 = (x[1] \oplus 1) \wedge y[1]$ . Then the  $j$ -th bit of maximum value between two inputs is defined as follows:

$$\begin{aligned} \max\{x, y\}[j] &= ((1 \oplus \text{com}(x, y)) \wedge x[j]) \oplus (\text{com}(x, y) \wedge y[j]) \\ &= x[j] \oplus (\text{com}(x, y) \wedge (x[j] \oplus y[j])). \end{aligned}$$

For the bit-sliced implementation, all the lengths are also expressed in a binary representation and we denote the maximum length of SNPs by  $\mu$ . It follows from the primitive circuits that we can evaluate the circuits homomorphically:

$$\left( \text{equal}(\mathbf{s}_i, \mathbf{s}'_i) \wedge (f_i \oplus f'_i \oplus 1) \oplus 1 \right) \wedge \max\{D_i, D'_i\}[j]$$

where  $\text{equal}(\mathbf{s}_i, \mathbf{s}'_i) = \wedge_{j=1}^{15} (\mathbf{s}_i[j] \oplus \mathbf{s}'_i[j] \oplus 1)$  has 1 if and only if  $\mathbf{s}_i, \mathbf{s}'_i$  are the same. Finally, one can decrypt the results and decode  $\ell(\mathcal{L})$  values from the output plaintext polynomials. More precisely, let  $\ell_{i,j}$  be the value at  $i$ -th slot which corresponds to the  $j$ -th bit. We see that  $\sum_{j=1}^{\mu} \ell_{i,j} \cdot 2^{j-1}$  is the approximate edit distance of SNV site  $i$ , hence we need only perform aggregation operations over them.



## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

### 5.2.3 Secure DNA Sequence Comparison with Integer-based Implementation

We explain how to evaluate the genomic algorithms homomorphically using the homomorphic encryption scheme over the integers. Since polynomial multiplication does not correspond to component-wise multiplication of the vectors, we have to consider another packing method instead of [LNV11]. Let us consider the polynomial-CRT packing method. Suppose that an integer  $m$  is chosen to be a power of two and let  $n = \phi(m) = m/2$ . The  $m$ -th cyclotomic polynomial  $\Phi_m(x)$  factors modulo 2 into a product of the same irreducible factors (*i.e.*,  $\Phi_m(x) = x^n + 1 = (x+1)^n \pmod{2}$ ), so we cannot apply batching technique with these parameters. We can instead do that if taking a prime  $t$  (not 2) such that the polynomial splits into the distinct factors modulo  $t$ , but the use of a different message space leads to change our primitive circuits.

As noted in Section 3.2.1, we see that for  $x, y \in \{0, 1\}$ , the following properties hold:  $x \oplus y = (x - y)^2$  and  $x \wedge y = x \cdot y$  where  $-$  and  $\cdot$  are arithmetic operations over integers. From these observations, we can amend the evaluation circuit for the edit distance as follows: We note that for  $\mu$ -bit integer  $x$  and  $y$ , the comparison circuit  $\text{com}(x, y) = c_\mu$  can be expressed as

$$c_j = (1 - x[j]) \cdot y[j] + (1 - (x[j] - y[j])^2) \cdot c_{j-1}.$$

for  $j \geq 2$  with  $c_1 = (1 - x[1]) \cdot y[1]$ . Since it is available to compute on large integer inputs, the maximum value is defined by

$$\max\{x, y\} = (1 - \text{com}(x, y)) \cdot x + \text{com}(x, y) \cdot y = x + \text{com}(x, y) \cdot (y - x).$$

Using these circuits, we compute the ciphertext given by the homomorphic operations

$$\left(1 + \text{equal}(\mathbf{s}_i, \mathbf{s}'_i) \cdot ((f_i - f'_i)^2 - 1)\right) \cdot \max\{D_i, D'_i\}$$

where  $\text{equal}(\mathbf{s}_i, \mathbf{s}'_i) = \prod_{j=1}^{15} (1 - (\mathbf{s}_i[j] - \mathbf{s}'_i[j])^2)$ . Then we get the encryp-

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

tions of the approximate Edit distance result of SNV  $i$ .

### 5.2.4 Implementation

In this section, we explain how to set the parameters for homomorphic evaluations and present our experimental results. We used BGV scheme with Shoup-Halevi's HE library [HS13] (called HELib). Our experiments with BGV were performed on a Linux machine with an Intel Xeon 2.67 GHz processor. We also implemented YASHE scheme with ARITH library in C. The measurements were done in an Intel Core 3.60GHz, running 64-bit Windows 7. The dataset consists of two individual genomes randomly selected from Personal Genome Project.

**How to Set Parameters.** The security of BGV relies on the hardness of the RLWE assumption. Similarly, YASHE is provably secure in the sense of IND-CPA under the RLWE assumption and DSPR assumption. The main difference between the schemes is that BGV uses an odd integer  $m$  while YASHE chooses  $m$  to be a power-of-two with a prime integer  $q$  such that  $q \equiv 1 \pmod{m}$ . In [LPR10], it was shown that the hardness of RLWE with the cyclotomic polynomial  $\Phi_m(x) = x^{\phi(m)} + 1$  can be established by a quantum reduction to shortest vector problems in ideal lattices. This means that YASHE is believed to be secure as long as the lattice problems are hard to solve. In YASHE scheme, we look for the parameter  $t \neq 2$  which maximizes the number of slots we can handle in one go. We fix the word  $\omega = 2^{128}$  for the evaluation key and the standard deviation  $\sigma = 8$  for the error distribution  $\chi_{err}$ . Since we can estimate the size of noise during homomorphic operations, we get the lower bound on  $q$  to ensure the correctness. We also have maximal values of  $q$  to ensure the desired security using the results of [LN14], so that we can have more loose bound than that from LP's method. Then we set  $m$  as a power-of-two to get a non-trivial interval for  $q$  and then select a smallest  $q$  in this interval.

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

Table 5.2 presents the parameter setting and Table 5.3 shows performance results for secure DNA sequence comparison using **BGV** and **YASHE**. All the parameters provide 80-bit security level. We give the plaintext modulus  $t$ , the size of the ciphertext modulus  $q$ , the lattice dimension  $n = \phi(m)$ , and the number of plaintext slots  $\ell$ . We also give the circuit depth  $L$  so that HE scheme can correctly evaluate such a computation on encrypted data. In particular, it can be considered as the number of ciphertext moduli in the **BGV** scheme. We consider the ciphertext size in kBytes for a set of parameters. The last columns give the timings for the key generation, encryption, evaluation and decryption.

We evaluated the performance with the input data of different sizes  $5K$  and  $10K$ . We implemented the comparison circuit with the same method as described in [CKL15, Lemma 1] in order to reduce the circuit depth over encryption.

As discussed in [KL15], given the parameter  $L$ , we obtain the approximate size of ciphertext modulus as  $\log_2 q \approx 43 + 18 \cdot (L - 2)$  for **BGV** when using  $t = 2$  and  $R = \mathbb{Z}[x]/(\Phi_{8191}(x))$ . Since it should support  $L = 7$  or  $8$  to correctly evaluate approximate edit distance algorithm, we use the modulus  $q$  around 130 to 150. On the other hand, the size of the parameter  $q$  in **YASHE** should be strictly larger than  $2L \log_2(nt) \approx 52L$  with  $t = 2^9$  and  $R = \mathbb{Z}[x]/(x^{8192} + 1)$ . So we used a 384-bit prime  $q$  such that  $q \equiv 1 \pmod{2^{14}}$ .

In the implementation of **YASHE** scheme, computing the inverse of  $f$  modulo  $q$  turns out to be the most-time consuming part of the key-generation, which runs in around 128.34 seconds. In total, it takes about 130.59 seconds to generate the public key, secret key and evaluation keys, while the key generation of the **BGV** scheme takes about 3.41 seconds in order to support 8 levels.

There is also quite a big gap between the two schemes in timings for a multiplication of ciphertexts: **BGV** takes around 0.07 seconds, while **YASHE** takes around 1.75 seconds (including the key switching step) under the parameter settings used in our experiments. For the efficiency of the **YASHE**

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

scheme, we might avoid a costly key switching step during the homomorphic multiplication; however, it supports a limited number of homomorphic multiplications without the key switching step. This follows since the noise grows exponentially with the multiplicative depth through such consecutive operations. One alternative is to use a *hybrid approach*, in which we leave out key switching in certain places but do it in others using the evaluation key with a power of the secret key so that one can keep the ciphertext noise small for correct decryption. As a result, polynomial multiplication modulo  $x^n + 1$  takes about 0.64 seconds, but it is still slower than that in BGV. As expected, BGV is faster than YASHE to evaluate the genomic algorithms for DNA sequence comparison.

Table 5.2: Parameters of implementations

Scheme	$t$	$\log_2 q$	$n$	$\ell$	$L$	$ \text{ct} $
BGV	2	150	8190	630	8	300kB
YASHE	8191	384	8192	4096	6	384kB

Table 5.3: Implementation results of approximate edit distance

Scheme	Size	KeyGen	Encrypt	Eval	Decrypt
BGV	5K	3.41s	16.98s	<b>40.86s</b>	2.97s
	10K		33.34s	<b>76.08s</b>	5.81s
YASHE	5K	130.59s	58.46s	<b>110.18s</b>	2.66s
	10K		116.61s	<b>245.04s</b>	5.07s

### 5.3 Secure Searching of Biomarkers

In this section, we describe how to securely search a set of biomarkers on encrypted genomes. The aim of this work is to create a homomorphic security system for searching a set of biomarkers on encrypted genomes. We propose

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

an efficient method to securely search a matching position with biomarker and extract the information of DNA sequences at the position without complicated computation such as comparison. The important feature of our method is to encode a genomic database as a single element of polynomial ring. The searching operation in a database is done by a single multiplication with the encoded query data. Finally the efficient extraction operation enables us to obtain the DNA sequences at the matching position which is moved to the constant term of output polynomial. This method has the advantage over previous methods in parameter size and computational complexity.

### 5.3.1 Privacy-Preserving Database Searching and Extraction

Throughout this section, we use the BGV-type homomorphic encryption scheme but we put the message in the upper bits similar to the YASHE scheme. That is, an ciphertext  $\mathbf{ct} = (c_0, c_1)$  of a message  $m$  has the decryption structure of the form  $c_0 + c_1 \cdot \mathbf{s} = (Q/t) \cdot m + e$  for some small polynomial  $e$ .

We assume that the integer  $m$  is a power of two so that  $n = m/2$  and  $\Phi_m(x) = x^n + 1$ . We adapt the conversion and modulus-switching techniques of [DM15]. The conversion algorithm changes an RLWE encryption of  $m = \sum_i m_i x^i$  into an LWE encryption of its constant term  $m_0$ , and the modulus switching reduces the ciphertext modulus  $Q$  down to  $q$  while preserving the message. We note that an LWE ciphertext is represented as a vector in  $\mathbb{Z}_q$  for some modulus  $q$ , and the decryption procedure is done by an inner product of the ciphertext and the secret key vector.

- **RLWE.Conv(ct)**: Given a ciphertext  $\mathbf{ct} = (c_0, c_1)$  with  $c_0 = \sum_i c_{0,i} x^i$  and  $c_1 = \sum_i c_{1,i} x^i$ , output the vector  $\mathbf{ct}' = (c_{0,0}, c_{1,0}, -c_{1,n-1}, \dots, -c_{1,1})$ .
- **LWE.ModSwitch(ct)**: Given a ciphertext  $\mathbf{ct} \in \mathbb{Z}_Q^{n+1}$ , output the vector  $\mathbf{ct}' \leftarrow \lfloor (q/Q) \cdot \mathbf{ct} \rfloor \in \mathbb{Z}_q^{n+1}$ .

As mentioned before, the decryption structure of an RLWE ciphertext

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

$\mathbf{ct} = (c_0, c_1)$  has the form  $c_0 + c_1 \cdot \mathbf{s} = (Q/t) \cdot m + e$  and its constant term is

$$c_{0,0} + c_{1,0}s_0 - \sum_{i=1}^{n-1} c_{1,n-i}s_i = (Q/t) \cdot m_0 + e_0.$$

It can be represented as an inner product of a vector

$$(c_{0,0}, c_{1,0}, -c_{1,n-1}, -c_{1,n-2}, \dots, -c_{1,1})$$

and the desired **LWE** secret key  $\vec{s} = (1, s_0, \dots, s_{n-1})$ . Hence the output of the conversion algorithm can be seen as an **LWE** encryption of  $m_0$ . It is also easy to check that if  $\mathbf{ct} \in \mathbb{Z}_Q^{n+1}$  satisfies  $\langle \mathbf{ct}, \vec{s} \rangle = (Q/t) \cdot m + e \pmod{Q}$ , then the output of **LWE.ModSwitch** algorithm satisfies  $\langle \mathbf{ct}', \vec{s} \rangle = (q/t) \cdot m + e' \pmod{q}$  for some  $e' \approx (q/Q) \cdot e$ . These techniques have been proposed for an efficient bootstrapping [DM15], but they will play totally different roles in our application. Finally an **LWE** ciphertext of modulus  $q$  can be decrypted by  $\vec{s}$  as follows.

- **LWE.Dec**( $\mathbf{ct}, \mathbf{sk}$ ): Given a ciphertext  $\mathbf{ct} \in \mathbb{Z}_q^{n+1}$ , output the value  $m \leftarrow \lfloor (t/q) \cdot \langle \mathbf{ct}, \vec{s} \rangle \rfloor_q$ .

If  $\langle \mathbf{ct}, \vec{s} \rangle = (q/t) \cdot m + e \pmod{q}$  for some small enough  $e$ , it returns the correct message  $m$  modulo  $t$ . More precisely, the decryption procedure works if  $|te/q| < 1/2$ .

Now let us consider a database of a set of  $N$  tuples. Each tuple consists of a set of pairs  $(d_i, \alpha_i)$  for  $i = 1, \dots, N$ , where  $d_i$  denotes a *data-tag* in the domain  $\{0, 1, \dots, \mathcal{T} - 1\}$  and  $\alpha_i$  represents the corresponding *value attribute* in a plaintext space  $\mathbb{Z}_t \setminus \{0\}$ . Note that all the tags should be distinct from each other. For instance, in the case of personal information database,  $\alpha_i$  may be the age of client whose identity number is  $d_i$ . Now consider the following search query: Given a *keyword-tag*  $d$  from a tag domain, select  $\alpha_i$  if there exists an index  $i$  such that  $d_i = d$ ; otherwise zero ( $\perp$ ).

The purpose of this section is to store the database and carry out a search query on the public cloud while the database information is securely

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

protected against malicious adversaries. Our main idea is the following encoding method of database suitable for the efficient computation of equality test and extraction:  $\text{DB}(x) = \sum_i \alpha_i x^{d_i} \in \mathbb{Z}_t[x]$ .

The client encrypts this polynomial with the RLWE public-key encryption scheme and stores the ciphertext  $\text{ct}_{\text{DB}}$  in the server. At the query phase, given a keyword-tag  $d$ , the client encrypts the monomial  $x^{-d}$  with the RGSW symmetric encryption scheme and sends the ciphertext  $\text{CT}_Q$  to the server. We assume that the RGSW encryption scheme has the same secret key  $\text{sk}$  as the one of RLWE encryption scheme.

**Remark 5.3.1.** Given an RGSW ciphertext  $\text{CT} \in \mathcal{R}_Q^{2d_g \times 2}$  and an RLWE ciphertext  $\text{ct} \in \mathcal{R}_Q^2$  output the vector  $\text{ct}' \leftarrow \text{CT}^T \cdot \text{WD}_{B_g}(\text{ct})$ . If  $\text{CT}$  and  $\text{ct}$  are RGSW and RLWE encryptions of  $m$  and  $m'$ , respectively, their multiplication  $\text{ct}'$  is a valid RLWE encryption of  $mm'$ . For convenience, we will denote  $\text{Hybrid.Mult}(\text{CT}, \text{ct})$  algorithm by  $\square$ , *i.e.*,

$$(\text{CT}, \text{ct}) \in \mathcal{R}_Q^{2d_g \times 2} \times \mathcal{R}_Q^2 \mapsto \text{CT} \square \text{ct} \in \mathcal{R}_Q^2.$$

Given two ciphertexts  $\text{CT}_Q \leftarrow \text{RGSW.Enc}(x^{-d})$ ,  $\text{ct}_{\text{DB}} \leftarrow \text{RLWE.Enc}(\text{DB}(x))$ , the server first performs their multiplication to obtain an ciphertext, denoted by  $\text{ct}_{\text{mult}} = \text{CT}_Q \square \text{ct}_{\text{DB}}$ . It follows from the previous section that  $\text{ct}_{\text{mult}}$  is a valid RLWE encryption of the polynomial

$$\text{DB}(x) \cdot x^{-d} = \sum_i \alpha_i x^{d_i - d} \in \mathcal{R}_t.$$

Since we use the cyclotomic polynomial  $\Phi_m(x) = x^n + 1$  of power-of-two degree, the polynomial ring  $\mathcal{R}$  has the property  $x^n = -1$ . Thus, for any tag  $d$ , the constant term of the polynomial  $\text{DB}(x) \cdot x^{-d}$  is  $\alpha_i$  if there is some index  $i$  satisfying  $d = d_i$ , otherwise zero.

Now the server applies the  $\text{RLWE.Conv}$  algorithm on  $\text{ct}_{\text{mult}}$  to compute an LWE encryption  $\text{ct}_{\text{conv}}$  of this constant term. This conversion procedure prevents the leakage of unqueried information and reduces the decryption cost.

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

In addition, the (optional) modulus-switching procedure can be considered to get a ciphertext  $\text{ct}_{\text{res}}$  with a smaller modulus size and reduce the communication cost. Finally the client decrypts this LWE ciphertext and gets the desired value  $\alpha_i$  or zero ( $\perp$ ). The Algorithm 4 summarizes the procedure of secure *search-and-extraction*.

---

### Algorithm 4 Procedure of secure search and extraction

---

- 1: Database encryption: The data owner encodes the genomic information as  $\text{DB}(x)$  and submits its encryption to the server:

$$\text{ct}_{\text{DB}} \leftarrow \text{RLWE.Enc}(\text{DB}(x)).$$

- 2: Query encryption: The user encodes the keyword-tag  $d$  and sends its encryption to the server:

$$\text{CT}_Q \leftarrow \text{RGSW.Enc}(x^{-d}).$$

- 3: Evaluation phase: The server computes their multiplication and carries out the conversion and modulus-switching operations:

$$\begin{aligned} \text{ct}_{\text{mult}} &\leftarrow \text{Hybrid.Mult}(\text{CT}_Q, \text{ct}_{\text{DB}}). \\ \text{ct}_{\text{conv}} &\leftarrow \text{RLWE.Convert}(\text{ct}_{\text{mult}}). \\ \text{ct}_{\text{res}} &\leftarrow \text{LWE.ModSwitch}(\text{ct}_{\text{conv}}). \end{aligned}$$

Return the resulting ciphertext  $\text{ct}_{\text{res}}$  to the user.

- 4: Decryption phase: The user decrypts the ciphertext with the secret key and gets the desired value:

$$\alpha \leftarrow \text{LWE.Dec}(\text{ct}_{\text{res}}).$$


---

Equality test has been traditionally considered difficult to perform on homomorphic encryption, because of its large circuit depth. However, our method is very efficient in parameter size and complexity since it requires only a single hybrid multiplication.

One limitation of this method is that the tags  $d_i$  should be bounded by ciphertext dimension  $n$  to construct the encoding polynomial  $\text{DB}(x)$ . Since the dimension  $n$  has a significant influence on the performance of HE scheme, too large value of  $n$  has an impractical impact on the performance. We will



## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

describe how to overcome this problem in the next section with respect to the application to genomic data.

### 5.3.2 Secure Searching of Biomarkers

We return to our main goal of our problem: secure searching of biomarkers by finding a query genome in database. We describe how to encode and encrypt the genotype information of VCF file in order to apply the privacy-preserving database searching and extraction.

The VCF file contains multiple genotype information lines, where each of them consists of a triple  $(\text{ch}_i, \text{pos}_i, \text{SNPs}_i)$  of chromosome number, position, and a sequence of SNP alleles. A chromosome identifier  $\text{ch}$  ranges from 1 to 22, X, and Y. A non-negative integer  $\text{pos}$  represents the reference position with the first base having position 1, and  $\text{SNPs}$  is a reference or alternate sequence in  $\{A, T, G, C\}^*$ . A query from user is also a triple of the same form and we aim to decide absence/presence of this biomarker in the database file.

We represent the sex chromosomes X and Y as 0 and 23, respectively. Then we define an encoding function  $\mathcal{E} : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$  by  $(\text{ch}, \text{pos}) \mapsto d = \text{ch} + 24 \cdot \text{pos}$ .

In the following, we describe how to encode the  $\text{SNPs}$ . For convenience we set the upper bound for the length of  $\text{SNPs}$ , so let  $n_{\text{SNP}}$  be the maximal number of reference (or alternate) alleles to be compared between the query genome and user genome in the target database. It is encoded as a binary string of length  $\ell_{\text{SNP}} = 2 \cdot n_{\text{SNP}} + 1$  with the same way of subsection 5.2.1. We convert it into an integer value, denoted by  $\alpha_i$ . For example, ‘GC’ is encoded as a bit string  $10||11||1||0 \dots 0$  of length  $\ell_{\text{SNP}}$ , where the single ‘1’ in the third block is padded. This bit string will be represented as an integer of 29.

Now consider the case that we wish to encode the reference and alternate alleles together. Let  $\alpha_i^{\text{ref}}$  and  $\alpha_i^{\text{alt}}$  denote the integer encodings of  $n_{\text{SNP}}$  reference alleles and  $n_{\text{SNP}}$  alternate alleles, respectively. Then we define an encoding  $\alpha_i$  by the concatenation of two encodings, *i.e.*,  $\alpha_i = 2^{\ell_{\text{SNP}}} \cdot \alpha_i^{\text{ref}} + \alpha_i^{\text{alt}}$ .

A database file is encoded as a set of pair  $(d_i, \alpha_i)$  for  $i = 1, \dots, N$  such that

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

$d_i = \mathcal{E}(\text{ch}_i, \text{pos}_i)$  and  $\alpha_i$  is the encoded integer of the  $i$ -th SNP allele string. Then the encodings  $d_i$  and  $\alpha_i$  are regarded as data-tag and value attribute, respectively. The data user constructs a polynomial  $\text{DB}(x) = \sum_k c_k x^k$  such that

$$c_k = \begin{cases} \alpha_i & \text{if } k = d_i \text{ for some } i, \\ \alpha \leftarrow \mathbb{Z}_t & \text{otherwise,} \end{cases}$$

and then encrypts the polynomial with the RLWE public-key encryption scheme as described above.

The query genes are also encoded as a pair of integers  $(d, \alpha)$ , however, we consider only the information of  $d$  is encrypted using the RGSW symmetric encryption scheme, that is, the user encrypts the monomial  $x^{-d}$ .

### 5.3.3 Optimization Techniques

As we mentioned before, the ring dimension  $n$  needs to be larger than the encoded integers  $d_i$ 's. However, the encoded integers  $d_i$  from VCF files have bits size about 32, while a dimension  $n$  with about  $11 \leq \log n \leq 16$  is considered appropriate for implementation of HE schemes to achieve both security and efficiency. Hence direct application of our method to the VCF file would yield an impractical result.

For compression of tag data and its re-randomization, we make the use of a pseudo random number generator  $H(\cdot)$  which transforms a tag  $d_i$  into a pair of two non-negative integers  $d_i^*$  and  $d_i^\dagger$  less than  $n$ . Our implementation adopts SHA-3 and extracts  $\log n = 11$  bits of the hashed value for each of two integers  $d_i^*$  and  $d_i^\dagger$ .

We construct two polynomials

$$\text{DB}^*(x) = \sum_k c_k^* x^k, \quad \text{DB}^\dagger(x) = \sum_k c_k^\dagger x^k$$

by the Algorithm 5. Note that for any  $1 \leq i \leq N$  and  $H(d_i) = (d_i^*, d_i^\dagger) \in \{0, \dots, n-1\}^2$ , the pair of constructed polynomials  $\text{DB}^*$  and  $\text{DB}^\dagger$  satisfy

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

---

### Algorithm 5 Encoding genomic data

---

```

1:  $c_{d_1^*}^* \leftarrow \alpha_1 \in \mathbb{Z}_q, c_{d_1^\dagger}^\dagger \leftarrow \alpha_1 - c_{d_1^*}^*$ 
2:  $d_1^* \in \mathcal{D}^*, d_1^\dagger \in \mathcal{D}^\dagger$ 
3: for  $i \in \{2, \dots, n\}$  do
4:   if  $d_i^* \notin \mathcal{D}^*$  and  $d_i^\dagger \notin \mathcal{D}^\dagger$  then
5:      $c_{d_i^*}^* \leftarrow \alpha_i \in \mathbb{Z}_q, c_{d_i^\dagger}^\dagger \leftarrow \alpha_i - c_{d_i^*}^*$ 
6:   else if  $d_i^* \in \mathcal{D}^*$  and  $d_i^\dagger \notin \mathcal{D}^\dagger$  then
7:      $c_{d_i^\dagger}^\dagger \leftarrow \alpha_i - c_{d_i^*}^*$ 
8:   else if  $d_i^* \notin \mathcal{D}^*$  and  $d_i^\dagger \in \mathcal{D}^\dagger$  then
9:      $c_{d_i^*}^* \leftarrow \alpha_i - c_{d_i^\dagger}^\dagger$ 
10:  end if
11:   $d_i^* \in \mathcal{D}^*, d_i^\dagger \in \mathcal{D}^\dagger$ 
12: end for
13: return  $\text{DB}^*(x) = \sum_k c_k^* x^k, \text{DB}^\dagger(x) = \sum_k c_k^\dagger x^k$ 

```

---

$\alpha_i = c_{d_i^*}^* + c_{d_i^\dagger}^\dagger$ . The procedure of database encoding for secure search of biomarkers is described in Algorithm 5.

Let  $\text{ct}_{\text{DB}}^*$  and  $\text{ct}_{\text{DB}}^\dagger$  denote the ciphertexts of the polynomials  $\text{DB}^*$  and  $\text{DB}^\dagger$ , respectively. Similarly, given the query encoding  $d$ , the user computes its randomized value  $H(d) = (d^*, d^\dagger)$  and encrypts the two polynomials  $x^{-d^*}$  and  $x^{-d^\dagger}$ . We denote the ciphertexts by  $\text{CT}_Q^*$  and  $\text{CT}_Q^\dagger$ . The server computes the hybrid multiplication to obtain the ciphertexts

$$\text{ct}_{\text{mult}}^* = \text{CT}_Q^* \boxplus \text{ct}_{\text{DB}}^*, \text{ct}_{\text{mult}}^\dagger = \text{CT}_Q^\dagger \boxplus \text{ct}_{\text{DB}}^\dagger.$$

Now let  $\text{ct}$  denote the ciphertext computed by the homomorphic addition between  $\text{ct}_{\text{mult}}^*$  and  $\text{ct}_{\text{mult}}^\dagger$ . Finally the server converts it into an LWE ciphertext and performs the modulus-switching procedure as described above. The Algorithm 6 describes the procedure of secure *search-and-extraction* using our proposed optimization techniques.

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

---

**Algorithm 6** Procedure of optimized secure search of biomarkers

---

- 1: Database encryption: The data owner encodes the genomic information as  $DB^*(x)$  and  $DB^\dagger(x)$ . Then the user submits the ciphertexts to the server:

$$\begin{aligned} ct_{DB}^* &\leftarrow \text{RLWE.Enc}(DB^*(x)), \\ ct_{DB}^\dagger &\leftarrow \text{RLWE.Enc}(DB^\dagger(x)). \end{aligned}$$

- 2: Query encryption: The user encodes the query as  $x^{-d^*}$  and  $x^{-d^\dagger}$ . Then the user sends the ciphertexts to the server:

$$\begin{aligned} CT_Q^* &\leftarrow \text{RGSW.Enc}(x^{-d^*}), \\ CT_Q^\dagger &\leftarrow \text{RGSW.Enc}(x^{-d^\dagger}). \end{aligned}$$

- 3: Evaluation phase: The server computes their multiplications:

$$ct_{\text{mult}}^* \leftarrow CT_Q^* \square ct_{DB}^*, \quad ct_{\text{mult}}^\dagger \leftarrow CT_Q^\dagger \square ct_{DB}^\dagger.$$

Let  $ct \leftarrow ct_{\text{mult}}^* + ct_{\text{mult}}^\dagger$ . The server converts it into an LWE ciphertext and performs modulus-switching operations:

$$\begin{aligned} ct_{\text{conv}} &\leftarrow \text{RLWE.Convert}(ct). \\ ct_{\text{res}} &\leftarrow \text{LWE.ModSwitch}(ct_{\text{conv}}). \end{aligned}$$

Return the resulting ciphertext  $ct_{\text{res}}$  to the user.

- 4: Decryption phase: The user decrypts the ciphertext with the secret key and gets the desired value:  $\alpha \leftarrow \text{LWE.Dec}(ct_{\text{res}})$ .
- 

### 5.3.4 Implementation

In this section, we explain how to set the parameters present our results using the optimization techniques. The dataset was randomly selected from Personal Genome Project. Our implementation is publicly available on [github](#) [KS16].

**How to Set Parameters.** Given the ciphertext modulus  $Q$ , it follows from the estimation of noise growth during evaluations [DM15] and decryption condition that we get the upper bound on the plaintext modulus  $t$  to ensure the correctness of decryption after computation. So we set  $t$  as the largest power-of-two integer less than the upper bound. If the encodings of

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

the allele strings are too large, we divide them into smaller integers so that each of them is smaller than  $t$ . Then we repeat the algorithm to construct the corresponding polynomials of each integer. On the other hand, the security of the underlying homomorphic encryption scheme relies on the hardness of the RLWE assumption. We derive a lower-bound on the ring dimension as  $n \geq \frac{\lambda+110}{7.2} \cdot \log Q$  to get  $\lambda$ -bit security level from the security analysis of [GHS12].

The use of variable type ‘*int32\_t*’ accelerates the speed of implementations and basic C++ std libraries, so we set  $Q = 2^{32}$  as the ciphertext modulus. We also set  $t = 2^{11}$  as the modulus parameter of the plaintext space to ensure the correctness for the output ciphertext. We take the following parameters for Gadget matrix  $\mathbf{G}$ :  $B_g = 128$  and  $d_g = 5$ , so that they satisfy the condition  $B_g^{d_g} \geq Q$ .

Each coefficient of the secret key  $\mathbf{sk}$  is chosen at random from  $\{0, \pm 1\}$  and we set 64 as the number of nonzero coefficients in the secret key. As in the work of [DM15], we considered the Gaussian distribution of standard deviation  $\sigma = 1.4$  to sample random error polynomials.

For the better efficiency of homomorphic multiplication, we also used the optimized library for complex FFT, *i.e.*, the *Fast Fourier Transform in the West* [FJ05]. That is, we use the complex primitive  $2n$ -th root of unity rather than a primitive root in a prime field of order  $Q$ . We measure a running time of 0.804 seconds to set up the FFT environment at dimension  $2n = 2^{12}$ . The key generation of two schemes takes about 0.247 ms in total.

Table 5.4 presents the time complexity and storage for the evaluation of secure searching of biomarkers. All the experiments were performed on a single Intel Core i5 running at 2.9 GHz processor. The chosen parameters provide at least 80 bits of security level.

## CHAPTER 5. SECURE GENOME ANALYSIS BASED ON HOMOMORPHIC ENCRYPTION

Table 5.4: Implementation results of secure searching of biomarkers

DB size	$n_{\text{SNP}}$	Complexity				Storage		
		Query-enc	DB-enc	Eval	Dec	Query	DB	Result
10K	2	3.247ms	0.003s	0.018s	0.004ms	160KB	3MB	0.75MB
	5		0.007s	0.039s	0.011ms		6MB	1.5MB
	10		0.014s	0.079s	0.027ms		12MB	3MB
100K	2		0.021s	0.111s	0.034ms		17MB	4.25MB
	5		0.042s	0.227s	0.064ms		34MB	8.5MB
	10		0.099s	0.454s	0.139ms		68MB	17MB
4M	2		0.745s	3.954s	1.171ms		593MB	148MB
	5		1.506s	7.911s	1.949ms		1185MB	296MB
	10		3.001s	15.442s	3.795ms		2370MB	593MB

## Chapter 6

# Conclusions

Our results to date scratch the surface of a HE-based general framework for private query evaluation and only implement general but relatively simple queries. These are just starting points on FHE-based private query processing and we need to implement a prototype of our solution working on top of a real DBMS, such as MySQL, which can be a final goal of our research.

We proposed a protocol for secure edit-distance computation over encrypted data. We also discussed how to privately perform genomic tests on encrypted genome data by using the approximate edit distance. The proposed protocols demonstrate the feasibility of securely outsourcing genomic data and edit distance computation in an untrusted cloud environment.

Finally we suggest an efficient method to securely search of biomarkers using hybrid GSW homomorphic encryption scheme. We came up with a solution to the secure outsourcing matching problem by using polynomial encoding and extraction of query SNP based on the multiplication of a RLWE-GSW ciphertext and an ordinary RLWE ciphertext. Our solution shows the development of cryptographic techniques can support real-world genomic data analysis.

Although FHE has a long way to go in its practical uses, the good news is that there have been significant improvements in HE schemes (*e.g.*, [CLT14, DM15, HS15]). We expect to have much faster performance in near future by

## CHAPTER 6. CONCLUSIONS

applying more efficient HE schemes to our current protocols instead of the BGV scheme and YASHE scheme.



# Bibliography

- [BGH<sup>+</sup>13] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J Wu. Private database queries using somewhat homomorphic encryption. In *International Conference on Applied Cryptography and Network Security*, pages 102–118. Springer, 2013.
- [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proc. of ITCS*, pages 309–325. ACM, 2012.
- [BLLN13] Joppe W Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and Coding*, pages 45–64. Springer, 2013.
- [Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.
- [BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS’11, pages 97–106. IEEE Computer Society, 2011.

## BIBLIOGRAPHY

- [CKK15] Jung Hee Cheon, Miran Kim, and Myungsun Kim. Search-and-compute on encrypted data. In *International Conference on Financial Cryptography and Data Security*, pages 142–159. Springer, 2015.
- [CKK16] Jung Hee Cheon, Miran Kim, and Myungsun Kim. Optimized search-and-compute circuits and their application to query evaluation on encrypted data. *IEEE Transactions on Information Forensics and Security*, 11(1):188–199, 2016.
- [CKL15] Jung Hee Cheon, Miran Kim, and Kristin Lauter. Homomorphic computation of edit distance. In *International Conference on Financial Cryptography and Data Security*, pages 194–212. Springer, 2015.
- [CLT14] Jean-Sébastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In *Public-Key Cryptography–PKC 2014*, pages 311–328. Springer, 2014.
- [DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.
- [DM15] Léo Ducas and Daniele Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology–EUROCRYPT 2015*, pages 617–640. Springer, 2015.
- [EN14] Yaniv Erlich and Arvind Narayanan. Routes for breaching and protecting genetic privacy. *Nature Reviews Genetics*, 15(6):409–421, 2014.
- [FJ05] Matteo Frigo and Steven G Johnson. The design and implementation of fftw3. *Proceedings of the IEEE*, 93(2):216–231, 2005.

## BIBLIOGRAPHY

- [FM91] Joan Feigenbaum and Michael Merritt. Open questions, talk abstracts, and summary of discussions. 1991.
- [Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. <http://crypto.stanford.edu/craig>.
- [GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.
- [GMG<sup>+</sup>13] Melissa Gymrek, Amy L McGuire, David Golan, Eran Halperin, and Yaniv Erlich. Identifying personal genomes by surname inference. *Science*, 339(6117):321–324, 2013.
- [GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology-CRYPTO 2013*, pages 75–92. Springer, 2013.
- [HAHT13] Mathias Humbert, Erman Ayday, Jean-Pierre Hubaux, and Amalio Telenti. Addressing the concerns of the lacks family: quantification of kin genomic privacy. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1141–1152. ACM, 2013.
- [HS13] Shai Halevi and Victor Shoup. Design and implementation of a homomorphic-encryption library. *IBM Research (Manuscript)*, 2013.
- [HS15] Shai Halevi and Victor Shoup. Bootstrapping for helib. In *Advances in Cryptology-EUROCRYPT 2015*, pages 641–670. Springer, 2015.

## BIBLIOGRAPHY

- [KL15] Miran Kim and Kristin Lauter. Private genome analysis through homomorphic encryption. *BMC medical informatics and decision making*, 15(Suppl 5):S3, 2015.
- [KS16] Miran Kim and Yongsoo Song. Implementation of secure searching of biomarkers, 2016. <http://github.com/amedonis/HybridHE>.
- [KSC17] Miran Kim, Yongsoo Song, and Jung Hee Cheon. Secure searching of biomarkers using hybrid gsw encryption scheme. preprint, 2017.
- [LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012*, pages 1219–1234. ACM, 2012.
- [LN14] Tancrede Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes fv and yashe. In *International Conference on Cryptology in Africa*, pages 318–335. Springer, 2014.
- [LNV11] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124. ACM, 2011.
- [LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology - EUROCRYPT 2010*, pages 1–23, 2010.
- [MS01] BA Malin and Latanya Sweeney. Inferring genotype from clinical phenotype through a knowledge based algorithm. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 41–52, 2001.

## BIBLIOGRAPHY

- [MS04] Bradley Malin and Latanya Sweeney. How (not) to protect genomic data privacy in a distributed network: using trail re-identification to evaluate and design anonymity protection systems. *Journal of biomedical informatics*, 37(3):179–192, 2004.
- [PRZB11] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.
- [RAD78] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.
- [S<sup>+</sup>01] Victor Shoup et al. Ntl: A library for doing number theory, 2001.
- [SAW13] Latanya Sweeney, Akua Abu, and Julia Winn. Identifying participants in the personal genome project by name. *Available at SSRN 2257732*, 2013.
- [SS11] Damien Stehlé and Ron Steinfeld. Making ntru as secure as worst-case problems over ideal lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 27–47. Springer, 2011.
- [SV14] Nigel P Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Designs, codes and cryptography*, 71(1):57–81, 2014.
- [TKMZ13] Stephen Tu, M Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing analytical queries over encrypted data. In *Proceedings of the VLDB Endowment*, volume 6, pages 289–300. VLDB Endowment, 2013.

## BIBLIOGRAPHY

- [WF74] Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.
- [ZDW<sup>+</sup>15] Y Zhang, W Dai, S Wang, M Kim, K Lauter, J Sakuma, H Xiong, and X Jiang. Secret: Secure editldistance computation over homomorphic encrypted data. In *Proceedings of the 5th Annual Translational Bioinformatics Conference Tokyo, Japan*, 2015.

## 국문초록

동형 암호는 복호화 과정을 거치지 않고 암호화된 상태에서 암호문끼리 연산을 가능하게 하는 암호 체계이다. 이론적으로는 동형 암호를 이용하여 임의의 함수를 암호화된 상태에서 계산할 수 있지만 상대적으로 큰 계산 비용으로 인해 아직까지 실용적이지 않으며 그에 대한 연구가 활발히 이루어지고 있지 않다.

본 학위 논문에서 동형 암호 기반 상등 또는 대소 비교, 그리고 정수 덧셈 연산에 대한 최적화된 서킷을 제안한다. 이를 바탕으로 동형 암호화된 데이터 베이스에서 안전하게 쿼리를 처리할 수 있는 방법에 대해 제안한다. 특히 키워드의 검색과 계산을 동시에 효율적으로 안전하게 처리할 수 있는 통합된 프로토콜을 설계한다.

한편 게놈 데이터 정보는 그 자체로 민감한 개인 정보이므로 이의 프라이버시를 보호하면서 클라우드 컴퓨팅 환경에서 분석 가능한 방법에 대한 연구가 새로운 화두로 떠오르고 있다. 암호화된 DNA 데이터 위에서 편집 거리(edit distance)를 계산할 수 있는 방법을 제안하고 이 기술을 개발하여 근사 편집 거리(approximate edit distance)를 분석할 수 있는 알고리즘을 제시한다. 마지막으로 암호화된 데이터 위에서 바이오마커(biomarker)를 찾을 수 있는 동형 암호 기반 보안 시스템을 제안한다. 이는 기존의 대소 비교와 같은 복잡한 연산 없이 바이오마커와 일치하는 위치를 검색하여 그 위치에 있는 DNA 염기 서열의 정보를 효율적으로 추출이 가능하다.

**주요어휘:** 동형암호, 정보 보호 쿼리 처리, SQL 쿼리, 편집거리, 근사 편집 거리, 바이오마커

**학번:** 2012-30071